REPC

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 1993	Final; September '89 - April '93

4. TITLE AND SUBTITLE
Flexible Work Group Methods in Apparel Manufacturing

5. FUNDING NUMBERS
DLA900-87-D-0018-0010

6. AUTHOR(S)
Charlotte Jacobs-Blecha, H. Don Ratliff, John J. Bartholdi, Richard Corey, Steve Nichols, Donald D. Eisenstein, Jon Lindbergh

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Georgia Institute of Technology
225 North Avenue
Atlanta, GA 30332

8. PERFORMING ORGANIZATION REPORT NUMBER
A-8496

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Logistics Agency
Manufacturing Engineering /Research Office
Cameron Station DLA-PR
Alexandria, VA 22304-6100

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
See DoDD 5230.24, "Distribution Statements on Technical Documents"

This document has been approved for public release and sale; its distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

In this project we have examined implementation methodologies for the modular manufacturing concept in the apparel industry. We looked at the traditional view of the modular group: a small group of operators, some of who are cross-trained on several operations in the group and can move around in the group to alleviate bottlenecks, and examined how to best coordinate the efforts of such a flexible work group. We also investigated the concept of TSS type work groups, showing a way to organize and operate such groups to attain maximum productivity. We developed an emulation of the modular concept, using off-the-shelf software which can be used for evaluative purposes. This is both a research tool and a decision support tool for the factory supervisor. We provide analytical results to show the methods we present are effective and efficient.

14. SUBJECT TERMS
Apparel Modular Manufacturing, Cellular Manufacturing, Flexible Manufacturing, Work Groups

15. NUMBER OF PAGES
62

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

DEFENSE LOGISTICS AGENCY
HEADQUARTERS
CAMERON STATION
ALEXANDRIA, VIRGINIA 22304-6100

12 AUG 1993

IN REPLY REFER TO AQPOT (Technical Enterprise Team)

SUBJECT: Report Documentation Page (Standard Form 298)

TO: Defense Technical Information Center
ATTN: DTIC-HDR

1. Enclosed are two copies each of a final report and SF 298, for the short-term research and development project entitled "Flexible Work Group Methods in Apparel Mannufacturing."

2. This report and accompanying documentation were submitted by Georgia Tech Research Institute under the base contract for Apparel Advanced Manufacturing Technology Demonstration (DLA900-87-D-0018).

3. Please call the undersigned or Mrs. Helen Kerlin, 46445, if you have any questions.

2 Encls

JULIE T. TSAO
Contracting Officer's
Technical Representative

SHORT TERM TASK
FINAL REPORT

FLEXIBLE WORK GROUP METHODS

IN APPAREL MANUFACTURING

Charlotte Jacobs-Blecha, Project Director
Computer Science and Information Technology Laboratory

John J. Bartholdi
Donald D. Eisenstein
H. Don Ratliff
School of Industrial and Systems Engineering

Richard Carey
Jon Lindbergh
Steve Nichols
Electronic Optics Laboratory

Georgia Institute of Technology

September, 1989 - April, 1993

Research Sponsored by:
U. S. DEFENSE LOGISTICS AGENCY
Contract No.: DLA900-87-D-0018-0010
Georgia Tech Project No.: A-8496

GEORGIA INSTITUTE OF TECHNOLOGY
A Unit of the University System of Georgia
Atlanta, Georgia 30332

Georgia Tech
RESEARCH INSTITUTE

SHORT TERM TASK
FINAL REPORT

FLEXIBLE WORK GROUP METHODS

IN APPAREL MANUFACTURING

Charlotte Jacobs-Blecha, Project Director
Computer Science and Information Technology Laboratory

John J. Bartholdi
Donald D. Eisenstein
H. Don Ratliff
School of Industrial and Systems Engineering

Richard Carey
Jon Lindbergh
Steve Nichols
Electronic Optics Laboratory

Georgia Institute of Technology

September, 1989 - April, 1993

Research Sponsored by:
U. S. DEFENSE LOGISTICS AGENCY
Contract No.: DLA900-87-D-0018-0010
Georgia Tech Project No.: A-8496

ABSTRACT

In this project we have examined implementation methodologies for the modular manufacturing concept in the apparel industry. We looked at the traditional view of the modular group: a small group of operators, some of who are cross-trained on several operations in the group and can move around in the group to alleviate bottlenecks, and examined how to best coordinate the efforts of such a flexible work group. We also investigated the concept of TSS type work groups, showing a way to organize and operate such groups to attain maximum productivity. We developed an emulation of the modular concept, using off-the-shelf software which can be used for evaluative purposes. This is both a research tool and a decision support tool for the factory supervisor. We provide analytical results to show the methods we present are effective and efficient.

TABLE OF CONTENTS

1.0 Introduction 1
1.1 Overview and Problem Definition 1
1.2 The IDEF Model 2
1.3 Scope of the Project 3

2.0 What Manufacturers Are Doing 3
2.1 Russell Corporation 3
2.2 Fashion Star 4
2.3 Oxford Slacks 4

3.0 Control Strategies for Flexible Work Groups 5
3.1 Introduction 5
3.2 Production Lines 6
3.3 Centralized Control of Worker-Machine Reassignments 8
3.3.1 Latency Time 8
3.3.2 1-Period Worker Assignments 8
3.4 T-Period Worker Reassignments 9
3.4.1 Improving Throughput With a T-Period Assignment Model 9
3.4.2 Measuring Throughput in a Multiperiod Model 10
3.5 A Heuristic for T-Period Worker Reassignments 11
3.5.1 Maxflow-Matching Heuristic 11
3.5.2 Minimum Cuts in the Maxflow Graph 12
3.5.3 A Maxflow-Matching Iteration 13

4.0 Production Lines That Balance Themselves 16
4.1 Introduction 16
4.2 Decentralized Control of a Production Line 18
4.3 TSS As a Pull System 23
4.4 Complicated Behavior 26
4.5 Related Work 33
4.6 Implementation of the TSS rules at the AMTC Demonstration Center 35
4.7 Conclusions 37

5.0 The Virtual Manufacturing Enterprise 39
5.1 Introduction 39
5.2 The VME Architecture 41
5.2.1 Description of Classes 41
5.2.2 Description of Source Files 44
5.3 How to Operate the VME 46
5.3.1 A Sample Simulation 46
5.3.2 The Preprocessor 50
5.3.3 Transition Table 50
5.3.4 Format of an Input Line 51
5.3.5 What the Arguments are For 53
5.3.6 A Quick Glossary of Terms 54

5.4. The VME Completion 54

6.0 Deliverable Demonstrations of the TSS line 55
6.1 Demonstration at the GTRI AAMTD site 55
6.2 Demonstration At An Apparel Manufacturing Site 55
7.0 Summary and Conclusions 57
8.0 References 59
Appendix A 60

ACKNOWLEDGMENTS

We offer our gratitude to Phil Williams of Oxford Slacks, Dave Howell of Russell, Co. and Andy Cooper of Fashion Star, Ind. for their assistance in our visits to their respective manufacturing facilities.

We thank Lyonia Bunimovitch for helpful technical discussions and Leonard J. Egan and Frank Guest of Americas 21st for showing us TSS in operation. We also appreciate support provided by the Defense Logistics Agency, grant #DLA900-87-D-0018-0010 (all researchers); the National Science Foundation, grant #DDM-9101581 (Bartholdi); the Office of Naval Research, grant #N00014-89J-1571 (Bartholdi and Ratliff); and IBM (Eisenstein).

1.0 Introduction

1.1 Overview and Problem Definition

In the apparel industry, Flexible Work Groups (FWGs)* are teams of workers cross-trained in several operations which carry out entire assembly processes and are compensated as a group rather than individually. The emphasis is on group effort and employee involvement, quality at the source, and short throughput time. This concept is being used in a number of production areas similar to apparel, such as shoe and curtain manufacture. Exploration of this manufacturing method is ongoing in the apparel industry, and many makers have expressed a strong interest in the concept.

In addition to the impact on productivity, there is likely to be an equally important impact on quality. People working together as a team are likely to be more consistent in the accuracy with which they perform their jobs. In this setting, there is the opportunity for real-time feedback as to fabric and sewing defects before the garment is completely assembled. This translates into raw material savings as well as a reduction in required production repetitions. In addition, the team members are apt to establish more pride in their work and motivation to assemble defect-free garments in their group. Producing high quality, defect-free garments is necessary to gain consumer confidence and to increase the competitiveness of U.S. apparel makers with overseas manufacturers.

A third benefit of the FWG philosophy may be a significant reduction in work-in-process (WIP) inventory, and the corresponding reduction in required plant floor space and carrying costs. The savings associated with reduced inventory are reflected along the entire inventory pipeline, including the supplier level. Furthermore, the physical reduction in WIP can greatly reduce flow congestion and thereby enhance material control.

The FWG concept brings a new set of challenges. For example, the question of how the manufacturing processes will actually be carried out becomes a much more complex one. This involves not only the layout of the equipment, but also a careful evaluation of which operations will be incorporated into the work module, which operators will work in the group, and which operator will perform which operation. In addition, the overall manufacturing process is likely

---

* These groups are also called Modular Manufacturing Groups. We will use both terms in this report.

to require a much more complex control strategy and tracking system for order progress to be updated.

The FWG concept brings social as well as technological challenges. First, the implementation of a FWG requires a cooperative and conscientious attitude from those persons working in the FWG. This may be brought about by the use of proper training and various incentive programs. Operator absenteeism becomes a problem when team operation depends upon everyone being present and contributing. In addition, the team concept requires a great deal more self-management, offering an even greater challenge to the workers involved. This is also likely to mean that plant managers must become more flexible and must set more realistic goals for meeting market conditions.

We make some basic assumptions concerning the enterprise in which the FWG will operate and what issues we will address. We assume the task (or tasks) to be accomplished by the group has been predetermined. We do not address the social issues. We focus only on the "how to" of improving the performance of the flexible work group.

1.2 The IDEF Model

This project fits into the IDEF models precisely as defined by Jayaraman and Malhotra (1992). This can be observed by examining nodes A53 and A532. In Node A53, Produce Garments, the manufacturing method represented by modular groups fits into the Cut Package at the Sew and Finish Garments node (A532). Further verification of this placement in the IDEF model presents itself in the link from Produce Garments to Grade and Sort Garments, labeled with "Finished Garments." The breakdown of Node A532 shows more of the modular concept. In node A5321, Control Sewing and Finishing Production, one sees the sewing and finishing assignments going out to nodes A5323 (Transport Garment Sub-Assemblies) and A 5324 (Process Garment Sub-Assemblies). From A5323 the garment subassemblies are moved from their storage area (whether it be a nearby table, rack, or another area of the plant) to the sewing line to be processed (node A5324).

Further examination of the breakdown of node A5324 shows even more detail. The modular line may be involved in all the sub-processes of the garment sub-assembly processing. These can be seen in nodes A53241, Set up sewing and finishing unit, A53242, perform sewing and finishing operations, A53243, inspect garment sub-assembly, and A53244, re-work garment sub-assembly. Note that these processes are connected sequentially. The actual performance of the sewing and finishing operations are preceded by the setting up of the sewing and finishing

unit, with the connection made by operator assignments. This key link is where we define the TSS rules as the operator assignments. Illustrations of these nodes from the IDEF model are given in Appendix A. Further details can be found in the technical report prepared by Jayaraman and Malhotra (1992).

It is also important to note that the organization of the sewing processes as defined in the IDEF model parallels that of the object-oriented structure defined in the architecture designed for our VME. This architecture is described in Section 5.

1.3 Scope of the Project

Our objective in this task order has been to investigate Flexible Work Groups and determine organization performance methodologies which will cause the FWG to perform more productively. We have accomplished this objective by investigating four topics. We have examined how the concept of FWG is being implemented in the apparel industry. From this, we have put forth two efforts to define methods for improving the performance of the FWG. The first effort deals with the dynamic operator scheduling problem. Following we will report on our research with the work groups that perform similarly to those marketed by Americas 21st The fourth topic is our Virtual Manufacturing Enterprise (VME) tool.

This report is organized as follows. Section 2.0 itemizes the trips we made to manufacturing facilities which were using various types of manufacturing methods. The dynamic worker assignment work is discussed in section 3.0, and the analysis based on the TSS concept in section 4.0. The VME development is described in section 5.0. Section 6.0 provides a summary of the work and some conclusions, Section 7.0 describes the demonstrations of the TSS line which have been performed and are planned for the near future, and references are cited in Section 8.0.

2.0 What Manufacturers Are Doing

2.1 Russell Corporation

Russell Corporation is located in Alexander City, Alabama. Dave Howell, Russell's Corporate Training Director, has been very helpful is helping us to understand how the work groups in Russell's plants are operated.

The teams are configured by fiat from management, with no formal analysis of how well their skills might complement the team. In addition, management suggests to the team captain

how some simple worker reassignments might be made during production. These are quite simple; they typically consist of not much more than moving person 1 to machine A when production there is behind schedule, and person 2 to machine B when it is behind. It seemed clear that our strategies could do better with insignificantly more effort.

Another point of interest at Russell included the continued use of bundles on the production floor. The casual observer would have a hard time discerning the difference from these modular teams and production using the progressive bundle system. However, the teams have increased productivity, although it is unclear why. It could be something as simple as a sense of camaraderie in the work force. For example, the operators were seen wearing team T-shirts with the team's logo. However, just having the ability to rescue an operation which has fallen behind in production may also be the underlying reason for the improvement.

2.2 Fashion Star

The project team visited Fashion Star, Inc., near Carrollton, Georgia. This company manufactures uniforms and specialty clothing on a contract basis. Their orders often consist of only a few garments (sometime even just one). This means that production must be very efficient. Changeovers must happen easily and the operators must be flexible in their work skills. There is a large variety of styles, fabrics, and alteration variations.

Currently, this company is using a Unit Production System on the sewing floor. There is very little work-in-process anywhere in this facility. Even the cutting area is clear because they cut only one garment at a time, due to the preciseness with which they construct their products, i.e., *cut-to-order*. In general, the entire plant seemed to be quite efficient.

Such efficiency in operation is a goal being reached for by most of the apparel manufacturing plants in the United States. We feel that the work being done in this research project can help many factories achieve that goal.

2.3 Oxford Slacks

Oxford Slacks is located in Monroe, Georgia. Phil Williams, the Division Staff Engineer was our host for this visit. We toured the entire plant at Monroe from operational planning the finished goods' distribution facility. However, our focus for the day was their experience with the progressive bundle system and attempts to use modular manufacturing.

A few years ago, much time and expense were invested in converting the Monroe plant to a modular manufacturing facility. Now, however, Oxford has deemed their efforts

unsuccessful, and has abandoned using the modular concept entirely. Problems at Oxford included groups that were too large, increased material handling time by operators, problems with compensation of the groups, and lack of enthusiasm from the workers, even after proper training. All of these could have been avoided if our study had been available to them.

Their current system uses progressive bundling, with huge amounts of work-in-process material on the sewing floor, in the cutting room, and in finishing/inspection. In addition, throughput time is far too long for today's dynamic market.

3.0 Control Strategies for Flexible Work Groups

3.1 Introduction

When a production line is labor-intensive the manufacturing firm faces numerous issues in how best to coordinate its work force. If the tools used for manufacturing are expensive, then multiple workers must share the tools on the production line for the operation to be economically viable. How best to coordinate multiple workers on a labor intensive production line is the focus of this section.

Figure 1 shows a prototypical manufacturing line consisting of machines and their associated buffers. These buffers hold goods that are waiting for the next operation. Each production unit starts in the buffer for machine one, and must go through each production step sequentially before a finished good is placed in the final buffer. We assume that workers cannot combine their efforts on one machine or production step.

The time required to complete a production step varies from machine to machine. For this reason, manning each machine with a worker results in an unbalanced line. That is, a fast machine at the front of the line can tend to build large buffers of work-in-process in front of slower work stations, while such fast stations at the end of the line can be starved for work awaiting product from slower machines. Instead, one can use fewer workers than machines in an attempt to better balance the line. Workers can be trained on more than one production step so that faster machines are operated less frequently than slower machines during the production cycle. Such a group of workers which move from one machine to another during production are an example of a *flexible work group*

If workers seldom move to new machines, excessive work-in-process can tend to build; however, moving workers too frequently might incur excessive costs for worker movement and machine set-up. In addition, the time required to complete a production step may be dependent on the worker operating the machine; complicating the coordination of the group.

Apparel manufacturing is an example of labor-intensive production. The machines are expensive, and thus the firm must use multiple workers on a production line. The skills required for the various production steps are similar enough that cross-training workers on multiple operations is successful. The industry has been pushed to lower their work-in-process, and flexible work groups are used to better balance the production lines and shorten the production cycle. Some apparel manufacturers specially train their workers and design their machines so that single unit batches can effectively be used as workers move from one machine to another after every production step is completed. Other manufacturers use larger batches, build more work-in-process, and move workers in the group less frequently.

We examine how best to coordinate the efforts of a flexible work group. How many workers should be used on a given production line, and how should their movement be scheduled? How much production capacity is gained by adding one more worker? Which workers should one choose for a given line and which machines should each operate? Is a complex algorithm to schedule workers necessary or can a simple scheduling algorithm perform just as well? This section addresses these and related questions.

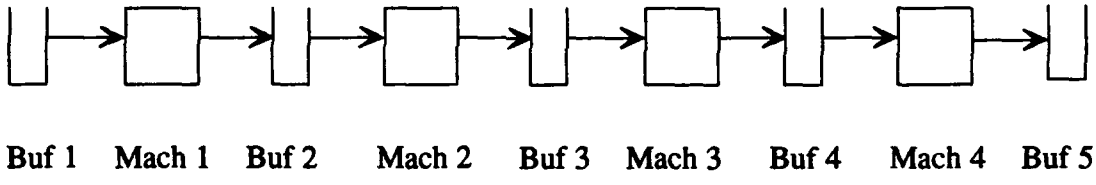


Figure 1. A four machine production line with asynchronous product transfer. Buffer 1 holds raw material and Buffer 5 holds finished goods.

3.2 Production Lines

We call each instance of the product an *item*. The average production rate of a production line is the number of items that have completed production divided by the time elapsed. We measure the effectiveness of our production lines by the average production rate. Accordingly we assume that there is no shortage of items to begin processing nor of space to put completed items.

Let the processing time of work station j be p_j, and let $P = \sum_j p_j$ be the total work content of the product. A line with n workers can produce, on average, no more than one item every P/n time units, a production rate of n/P items per time unit. If a line produces at a rate of n/P we say

that it *is balanced* because each worker, on average, contributes an equal amount of time P/n on each item produced.

Since only one worker can operate a work station at a time, the line cannot produce items any faster than the processing rate or any single work station. Thus the production rate can be no larger than $1/p_{max}$ where p_{max} is the longest processing time on any work station. We call this work station with the largest p_j the *bottleneck* work station, since its processing rate can restrict the production rate of the entire production line.

Figure 2 shows the production line where the work content of the product is depicted as a horizontal line of length P, and individual work stations correspond to intervals of the line. Figure 3 is a "displacement diagram" of the production line, where the Work content of the product is again depicted as a horizontal line of length P. The diagram also shows the relative position of each worker at a particular instant in time. Since only one worker can operate a work station at a time, only one worker can be depicted on the displacement diagram within the interval of any given work station. Initially we assume that processing times are deterministic and independent of the worker, so that during production each worker can be imagined to move along the line of work content at the same uniform speed.

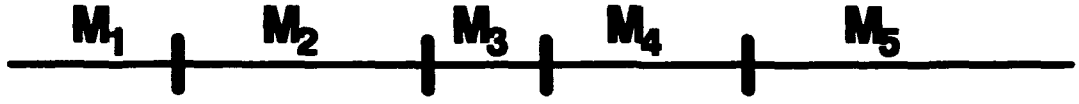


Figure 2. The production line depicted as a line of length P. Each work station corresponds to an interval of the line.

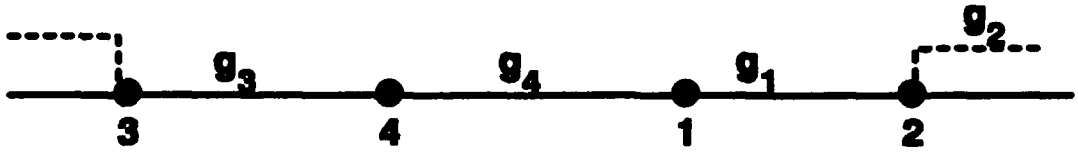


Figure 3. The displacement diagram with 4 workers. The g_i are the gaps between successive workers.

A worker's *predecessor* on the production line is the closest worker who is nearer the start of the line. If no other worker is nearer the start of the line, his predecessor is the worker nearest the end of the line. A worker's *successor* on the production line is the closest worker who is nearer the end of the line. If no other worker is nearer the end of the line, his successor is the worker nearest the start of the line.

We call the length of the interval of work content between worker i and his successor a *gap* and let g_i denote the length of this gap measured in units of processing time.

Lemma 1 $\sum_{i=1}^{n} g_i = P$ *throughout production.*

Proof: The gaps always form a partition of the work content. □

3.3 Centralized Control of Worker-Machine Reassignments

We consider centralized rules for making *synchronous* worker-machine reassignments in which the worker reassignments are made at predetermined times. Two variations of this model are considered–in the single time period or 1-period model the worker-machine assignment is fixed throughout the entire production cycle; in the multiperiod or T-period model the worker-machine assignments can change at the start of each of T time periods.

3.3.1 Latency Time

Consider a six-machine 1-period model in which each machine with its worker assignment can process a unit every 10 seconds. The time to handle the unit before and after production is considered negligible or included in the production times. The first unit is produced after one minute. In the course of a production day, the initial delay time or *latency* for the first unit to complete production is negligible. Thus to simplify throughput calculations we ignore the latency time of the first unit through the line. For T-period models we ignore the latency time of the first unit at the start of each time period. This will not introduce significant inaccuracies as long as the length of each time period is much greater than the latency time.

3.3.2 1-Period Worker Assignments

Consider the problem of assigning n workers to n machines in a 1-period model. The throughput of the production line is equal to the production rate of the slowest operation on the line. Thus we seek to find a worker assignment that maximizes the minimum production rate over all operations. Such an optimization problem is known as the "bottleneck matching" problem. A simple approach to solving the bottleneck matching problem is to solve a series of regular matching problems. We first attempt to find a feasible matching using only the largest production assignments of worker to machines. We continue to allow smaller and smaller production assignments until each machine can be assigned a worker. The resulting assignment is optimal for the 1-period model. The above approach is illustrative, but much more efficient procedures have been developed (see Chvatal, 1983 and Lawler, 1976).

		machin es		
		1	2	3
	a--	10	20	30
workers	b--	10	20	30
	c--	100	100	100

Figure 4. Worker Rates for 3 worker 3 machine problem

Figure 4 shows the number of units each worker can produce per hour for a 3 worker-3 machine instance. A solution to the bottleneck assignment assigns worker *c* to machine 1, *b* to 2, and a to 3, yielding *a* throughput of 20 units/hour.

3.4 T-Period Worker Reassignments

Reassigning workers throughout the production cycle can improve throughput over the fixed assignment–in fact we show that the throughput can be improved by an arbitrary amount. A flow model is also given that efficiently determines the throughput of a T-period assignment.

3.4.1 Improving Throughput With a T-Period Assignment Model

Figure 5 shows a 3 worker-3 machine instance where the rate of worker c is denoted by any arbitrarily large constant *M*, i.e., worker 3 is *much* faster than the other workers on the line. Consider a production cycle composed of 3 time periods, each time period of length one hour. Workers are only permitted to be reassigned at the start of each one hour time period. In this instance it is clear that throughput is maximized by spreading worker *c* over each machine. An optimal schedule is shown in Figure 6. Note how the fast worker *c* starts at machine 1 and then moves to machines 2 and 3 respectively. The successive reassignments allow the fast worker *c* to move the bulk of the units through the production line.

		machin es		
		1	2	3
	a--	1	1	1
workers	b--	1	1	1
	c--	M	M	M

Figure 5. Worker Rates for 3 worker 3 machine problem

		machines		
		1	2	3
	3--	a	b	c
time periods	2--	a	c	b
	1--	c	a	b

Figure 6. Optimal Worker Assignments

Figure 7 indicates the potential production capacity of each machine in each time period according to the worker rates specified by the optimal worker assignments of Figure 6. In this instance each machine is able to produce up to its capacity in each time period, and thus Figure 7 also shows the actual production flow of units. This production flow achieves a total production of $M + 2$ units for the 3 hour production cycle. A solution of the bottleneck assignment for the 1-period model yields a production of only 3 units for the 3 hour production cycle. Thus it is possible to get arbitrarily better production from a T-period model than the 1-period model.

	machin es		
	1	2	3
3--	1	1	M
time periods 2--	1	M	1
1--	M	1	1

Figure 7 Optimal Worker Assignment Rates and Production Flows

3.4.2 Measuring Throughput in a Multiperiod Model

Given a worker-machine assignment in a T-period model, one can determine the production of finished product by solving a maximum flow or "maxflow" problem (see Chvatal, 1983 and Nemhauser and Wolsey, 1988). The maxflow network has the structure shown in Figure 8, on the next page.

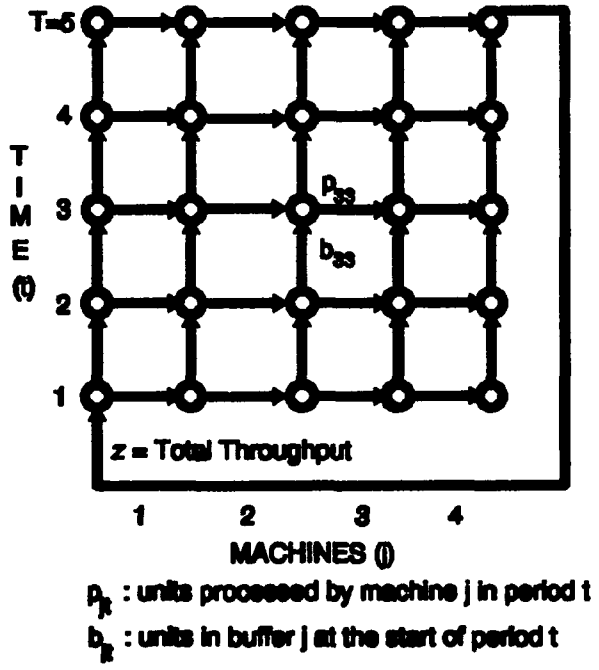


Figure 8. Maxflow for a 4 Machine-5 Period Model.

A unit of flow on the horizontal arcs represents a unit of production on a given machine in a given time period. The vertical arcs represent units that remain in the buffer of a machine

from one time period to the next. The capacity of a given horizontal arc corresponds to the production capacity of that machine for that time period. The capacity of such a machine is determined by the worker assigned to that machine in that time period. The capacities of vertical arcs are those of the corresponding buffers. The flow in the network is modeled with buffer 1 in time period 1 representing the source node and buffer $n + 1$ in the last time period being the sink node. The maximum throughput is then determined by solving for the maxflow in this network.

The nodes represent buffers; horizontal arcs represent production; vertical arcs represent work-in-process at the end of a planning period, the last time period being the sink node. The maximum throughput is then determined by solving for the maxflow in this network.

3.5 A Heuristic for T-Period Worker Reassignments

We present a heuristic for multiperiod models in which the buffer capacity is effectively infinite.

3.5.1 Maxflow-Matching Heuristic

In the maxflow model for measuring throughput, various schedules or worker-machine assignments over time affect only the capacities on the horizontal or machine production arcs of the maxflow. Our optimization problem is to find a worker schedule that creates the horizontal arc capacities in the maxflow model that allow the largest flow. We present a heuristic that iterates between solving the maxflow problem and finding an improved worker assignment.

3.5.2 Minimum Cuts in the Maxflow Graph

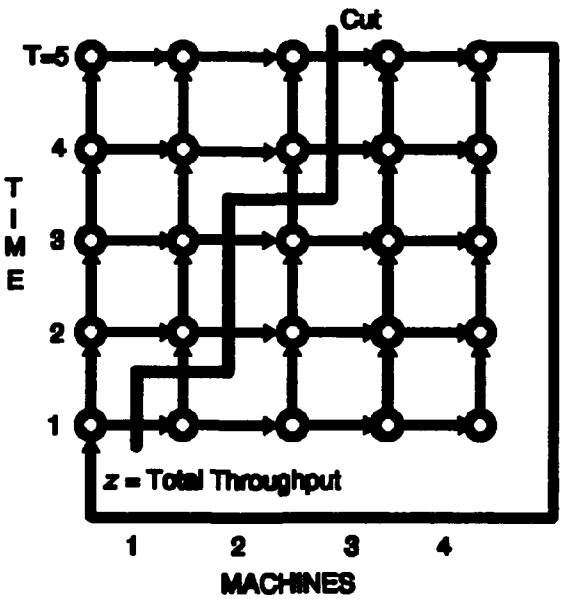


Figure 9. A Cut in a Maxflow Graph

An optimal solution to the maxflow problem also determines a minimum cut in the graph (see Chvatal, 1983 and Nemhauser and Wolsey, 1990). Figure 9 shows an example of a cut in a maxflow graph. When such a cut is of minimum capacity it provides some useful information to both a manager of the production line and to one seeking optimal worker schedules. The min cut defines a set of edges, machine-time period pairs, that form the bottleneck of the production line; that is, the capacity of at least one of the edges in the min cut must be increased if the production cycle is to produce any additional units within the production horizon T. For example, if the cut shown in Figure 9 is a minimum cut then it indicates that additional capacity on machine 4 in any time period will not improve the throughput of the line. In particular the throughput will not be improved unless the capacity of machine 3 in time periods 4 or 5, machine 2 in time periods 2 or 3, or machine 1 in time period 1 is increased. One should note that throughput is not guaranteed to improve if such capacities are increased, but that the throughput will definitely not improve unless the aggregate capacity of this particular set of arcs is increased. We are not assured of an improvement in throughput with an increase in the capacity of the min cut because multiple min cuts of the same aggregate capacity may exist.

In the uncapacitated model minimum cuts have a special structure. As the time period increases the machine number in the cut will monotonically increase. This is because one cannot have an infinite capacity vertical arc in the minimum cut, which is the case if the cut "doubled back". Thus we conclude the following.

Theorem *For the uncapacitated T-period model the production bottleneck machine will remain the same or move down the production line as the time period increases.*

Proof: Suppose on the contrary that machines a and b are the bottleneck machines in time periods $t + 1$ and t respectively, $a < b$ Thus the capacity of the min cut includes the capacity of the vertical arc representing the buffer of machine $a + 1$ into time period $t + 1$. Since the model is uncapacitated this implies that the capacity of the min cut is infinite. But since all machine capacities in all time periods are finite a cut of finite capacity is achieved by a simple vertical cut through any machine in all time periods.

3.5.3 A Maxflow-Matching Iteration

Consider any worker schedule. After solving the maxflow problem we have isolate a minimum cut whose capacity C equals that of the total production of the line. We want to find a new worker-machine assignment that makes the minimum capacity of all cuts greater than C. If we are successful in finding such a reassignment then the throughput will improve. The

heuristic attempts to find such a reassignment by focusing its search on one time period at a time.

Consider any single time period t in which we seek a worker reassignment that will improve throughput. To do this we need to know the impact of a reassignment for each machine j in time period t; that is, we must understand the effect of a worker assignment in time t on the cuts of the maxflow graph. We first remove the assignments for time t and then determine the minimum capacity cut through each machine or horizontal arc for time period t – denoting the capacity of such a min cut though machine j as c_{jt}. Thus we now know that the throughput will improve if the capacity we assign to each machine j in time period t plus the value c_{jt} exceeds

We can easily look for the existence of such a reassignment for time t by looking for a bipartite matching in a graph with nodes corresponding to workers matched with nodes for each machine. An arc is included from a given worker to a machine j if and only if the resulting capacity of the assignment plus c_{jt} exceeds C.

Thus we now just look for any feasible matching in this bipartite graph for time period t if one is found an improvement in throughput will result. If a matching is not found then we are assured that there is no possible reassignment of workers in just time period t that can improve throughput. We then can repeat the procedure for other time periods. After no more improvements can be found for any time period the heuristic terminates. Such a procedure is not guaranteed to be optimal, since an improvement in throughput may be obtainable by changing the assignments simultaneously in more than one time period. A more formal presentation of the algorithm is given in Figure 10.

To calculate each c_{jt} we solve a maxflow with each machine's capacity in time period t set to infinity, except for that of machine j, which we set to zero. The resulting maxflow yields the value c_{jt}.

To update the capacity of the min cut we could resolve the maxflow with the new assignment from the matching. A more efficient procedure takes the minimum over all machines at time T of c_{jt} plus the capacity of the new assignment for machine j.

Let r_{ij} = Capacity of Worker i on Machine j for one period; Make any Initial Worker Assignment or Null Assignment; Solve Maxflow, SET C = Capacity of Min Cut; SET t = 1;

WHILE t ≤ Number_of_Time_Periods DO SET Assignment of Workers at Time t to Null Assignment;

FOR j = 1 to Number_of_Machines DO

SET c_{jt} = Min Cut Capacity Through Machine j at time t;

ENDFOR

Create Bipartite (WORKER)x(MACHINE) Matching;

Include Edge(i, j) if and only if $r_{ij} + c_{jt} > C$;

IF Feasible Matching Found THEN

Update Assignment for Time Period t with Matching;

Update C = Capacity of Min Cut;

SET t = 1;

ELSE

SET t = t+ 1;

ENDIF

ENDWHILE

Figure 10. Maxflow-Matching Heuristic.

3.5.4 A Sample Problem

Figure 11 shows the rate matrix for an hour of production for an 8 worker-8 machine instance.

		machines							
		1	2	3	4	5	6	7	8
workers	a	10	10	10	10	10	10	10	10
	b	1	4	3	8	0	22	10	11
	c	8	8	9	1	1	3	3	4
	d	12	13	0	11	11	16	12	0
	e	3	2	9	2	5	6	9	1
	f	1	25	3	1	3	4	7	1
	g	1	0	1	2	5	6	1	1
	h	12	1	0	11	5	16	1	1

Figure 11. Worker Rates for *8 worker 8 machine* Example

The heuristic was implemented on this sample problem using 10 one-hour time periods. The results are shown in Figure 12. The straight line shows the throughput over time achieved by the optimal bottleneck solution for the 1-period or fixed worker assignment problem. If we instead allow workers to change assignments in a 10-period model we achieve superior results by the end of the production cycle. It is interesting to note that the T-period assignment was not always superior to the 1-period model until later in the production cycle. This is because the T-period model was able to utilize its buffers and, in effect, could position itself for greater throughput later in the cycle.

4.0 Production Lines That Balance Themselves

4.1 Introduction

Traditional means of organizing a production line, such as a classical assembly line, are inflexible. In a classical assembly line, workers are assigned fixed work stations and the station with the greatest work content determines the production rate. Realistically, there are only two ways to change the production rate: either change the number of shifts, or else redistribute the tasks, tools, and parts over different stations. The first allows only coarse adjustments and the second is expensive and disruptive. This inflexibility is partly due to the way in which traditional production is organized, where a centralized authority designs a globally coordinated pattern of material movement and task execution that is then rigidly followed by all workers.

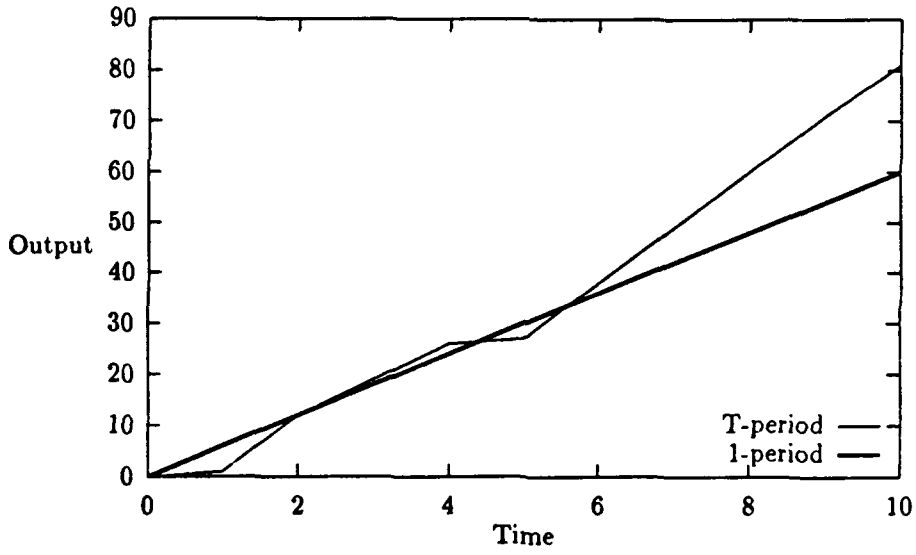


Figure 12:Heuristic T-period vs. Optimal 1-period Results of Sample Problem

It is particularly important that production systems be flexible when products have extreme seasonalities or short life-cycles, such as in the apparel industry. To increase flexibility of production, there has recently been introduced into the apparel industry a variation of the assembly line in which there are fewer workers than stations and workers walk to adjacent stations to continue work on an item. Control of the line is decentralized: each worker independently follows a simple rule that determines what to do next. This idea has been commercialized by Aisin Seiki Co., Ltd., a subsidiary of Toyota, and named the "Toyota Sewn Products Management System," or TSS[1]. It is marketed in the western hemisphere by Americas 21st

We shall prove that, when a TSS line is configured as a "pull" system, then, during the natural operation of the line, the work content of the product will be spontaneously reallocated among the workers to balance the line without conscious intention or management intervention. Furthermore, when no station has "too much" work, then the line achieves the highest production rate possible for the given workers. This capacity for self-organization allows management to fine-tune the production rate by simply changing the number of workers on the line, which in turn elicits a spontaneous reallocation of work.

4.2 Decentralized Control of a Production Line

Call each instance of the product an *item* and consider a flow line in which each item requires identical processing on the same sequence of m work stations, as in Figure 13. A station can process at most one item at a time, and exactly one worker is required to accomplish the processing. (This type of line is typical of the apparel industry, where each work station is generally a sophisticated sewing machine through which an item must be guided by a single worker.)

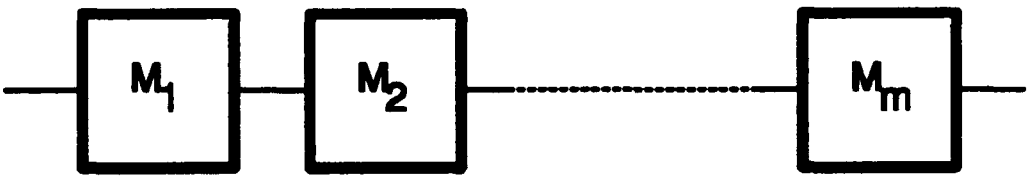


Figure 13. A simple flow line in which each item requires identical processing on the same sequence of work stations.

[1] A registered trademark of Aisin Seiki Co., Ltd.

We assume that each item requires the same total amount of processing, according to some work standard, and we normalize that total to one "time unit." Let the processing requirement at station j be p_j, a fixed percentage of the total standard work content of the product. For convenience, define $p_0 = 0$.

Unlike most other production models, ours includes specific representation of the human workers on the production line. Our model formalizes the notion that some workers are faster than others: The skill of each worker is summarized by his "velocity" $v_i > 0$, where the time it takes worker i to complete the work at station j is p_j/v_i.

Of course this model is not perfect (no model of people is likely to be). The obvious weakness is the assumption that the velocity of each worker is uniform over all tasks. (For example, this assumption is grossly violated at robotic machines on which a worker simply loads the item for processing and then removes it when done, so the processing time at such a station is largely independent of the attendant worker.) Nevertheless, each of the three production managers we interviewed agreed that some workers are generally faster than others and that our model was a reasonable approximation of this, especially if velocity is interpreted as a measure not just of skill but also of vigor, motivation, and enthusiasm. The managers also pointed out that this model seems natural in the apparel industry, where worker skill levels are monitored and expressed as a percentage of work standards.

One way of evaluating the productive capacity of a line is by estimating its maximum long-term average production rate during the manufacturing of a single product. In our model, no worker can work at more than one station at a time so that worker i cannot produce more than vi items per time unit; consequently *no* way of organizing the workers can achieve an average production rate that exceeds $P = \sum_{i=1}^{n} v_i$ items per time unit (where n is the number of workers); and this rate is achievable only if no worker is ever blocked. We will show that, when properly configured, a TSS line will spontaneously allocate work content among the workers to achieve this ideal.

The TSS line functions as a sort of "bucket brigade" in which workers move down the line, each devoted to a single item. When the last worker completes his item, he releases it, moves back up the line, and takes over the work of his predecessor, who in turn preempts his predecessor, and so on until the first worker, after having been preempted, introduces a new item to the line. This behavior is realized by requiring each worker to independently follow these rules:

TSS Rule (forward part) Remain devoted to a single item, and process it on successive work stations, queuing before a busy work station if necessary. If you complete processing the item or if you are preempted by another worker, then release the item and begin to follow the Backward Part.

TSS Rule (backward part) Walk back toward the beginning of the line until you encounter an item. If necessary, preempt the worker with that item. Begin following the Forward Part.

Let the number of workers n and their velocities v_i be fixed. Then the production rate of a TSS line is determined by the sequence of the workers on the line (which by the logic of the TSS Rule remains fixed), and by the initial positions of the workers when production is begun.

It is difficult to visualize the behavior of workers on a TSS line because they operate asynchronously. It is helpful to view the line as a dynamic system (see, for example, Devaney, 1989). The state of the system at any time t is given by the vector of worker positions $\mathbf{x}(t) = (x_1(t), \ldots, x_n(t))$, where the position of worker i is expressed by the fraction $x_i(t)$ of work completed on his item by time t, as illustrated in Figure 14.

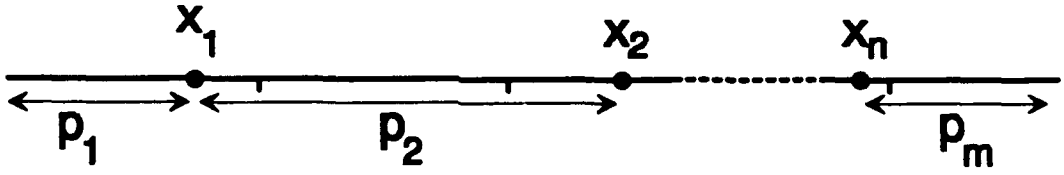


Figure 14. The standard work content of the product is represented as a line segment normalized to length 1. The position of worker i is given by x_i, the cumulative fraction of work content completed on his item.

The state space of the system is a subset of the unit hypercube, as illustrated in Figure 15. Note that $x_1 \leq x_2 < < x_n$ because the TSS Rule does not allow workers to pass one another. The saw-toothed edge of the feasible region arises because no more than one worker can use a station at a time.

The vector-valued function $\mathbf{x}(t)$ expresses the dynamics of the TSS system, but unfortunately, $\mathbf{x}(t)$ is difficult to describe directly. Accordingly, we make an important modeling assumption that is consistent with the real lines we have seen: that the time to walk back and preempt a worker is small. Therefore we can imagine that when the last worker finishes an item, then–*at the same instant*–worker n preempts worker n–1, who preempts worker n–2, . . ., who preempts worker 1, who introduces a new item into the system. We say that the line *resets* at

such an instant, and we call the time between resets a *phase*. This simplification frees us from worry about the details of $\mathbf{x}(t)$: We can restrict our attention to the sequence $\{\mathbf{x}^0, \mathbf{x}^1, \mathbf{x}^2, \ldots\}$ of worker positions at those instants when the line resets[2]. Continuing the metaphor of a dynamic system, we call such a sequence the *orbit* of worker positions beginning at $\mathbf{x}^0$.

The duration of a phase is the time between successive completions of items and is therefore the *cycle time* of the line. In a TSS line, successive cycle times can differ because phases are not all of the same duration: If the position of the last worker at the beginning of phase p is x_n^p then phase p will be of duration $(1 - x_n^p)v_n$.

Let x^p be the positions of the workers at the start of phase p. Then by definition $x_1^p = 0$ and for convenience we define $x_{n+1}^p = 1$. If $x_i \in (\sum_{j=0}^{k-1} p_j, \sum_{j=0}^{k} p_j)$ then worker i is busy at station k and, because workers are not allowed to share stations, there can be no other x_i with a value in this interval. Let $b(x_i)$ and $e(x_i)$ be the cumulative fractions of work content at the beginning and end of the station currently occupied by worker i. If worker i has just completed work at station k, so that $x_i = \sum_{j=0}^{k} p_j$ then define $b(x_i) = e(x_i) = x_i$.

The vector x^p can be interpreted as suggesting an allocation of work, with the interval of work content $[x_i^p, x_{i+1}^p]$ assigned to worker i. Define the *gap* g_i between worker i and $i+1$ as the clock time required for worker i to complete his imputed allocation of work, as follows.

$$g_i^p = \begin{cases} \dfrac{(x_{i+1}^p - x_i^p)}{v_i} + \max\left[0 \text{ and } \dfrac{(e(x_{i+1}^p) - x_{i+1}^p)}{v_{i+1}} - \dfrac{(b(x_{i+1}^p) - x_i^p)}{v_i}\right] & \text{if worker i+1 is not currently blocked.} \\ \max\left[\dfrac{(x_{i+1} - x_i)}{v_i} \text{ and } g_{i+1}\right] & \text{if worker i+1 is blocked.} \end{cases} \quad (1)$$

---

[2] Such a restriction is called a *Poincare section* of the dynamic system and is a standard technique for analyzing complicated dynamics.

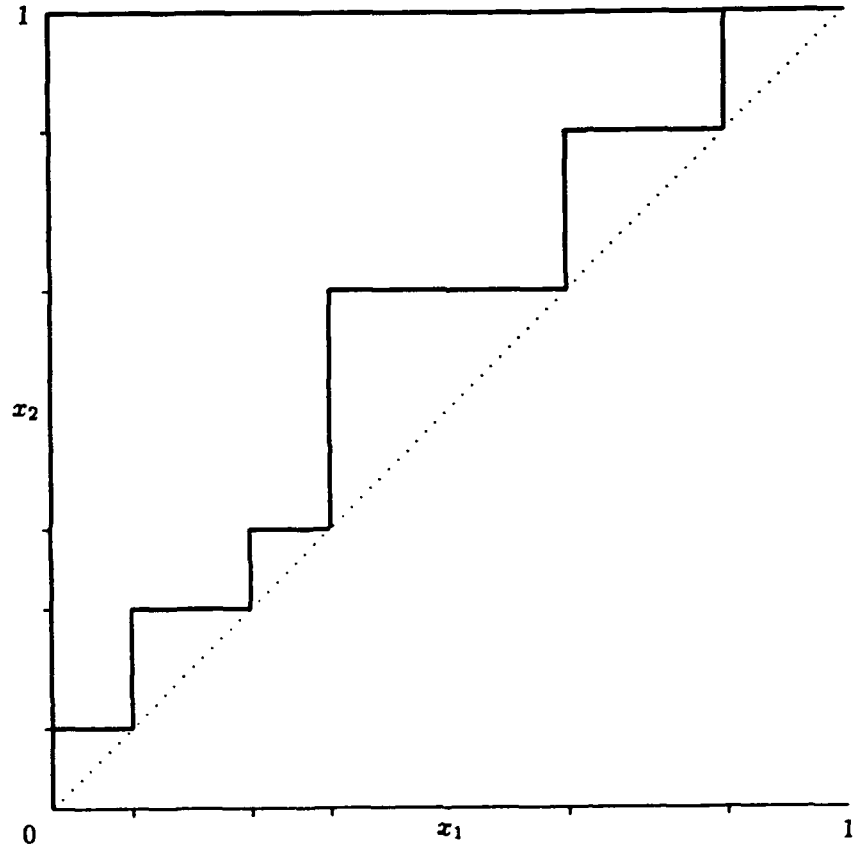


Figure 15 The phase space of a TSS line with two workers, whose positions are given by (x_1, x_2), is that portion of the unit square on or above the saw-tooth line. (The tick marks on the axes correspond to the partition of work among the stations.)

The first part of expression 1 consists of two terms: the first is the clock time required for worker i to reach the position of his successor $i+1$ if worker i is not blocked; and the second term is the delay if worker i is blocked, in which case i reaches $i+1$'s machine at time $(b(x_{i+1}^p) - x_i^p)v_{i+1}$ but $i+1$ does not finish there until time $(e(x_{i+1}^p) - x_{i+1}^p)v_{i+1}$. The second part of expression 1 defines the gap in "degenerate" cases, when several successive workers block each other.

For worker positions x^p we say that $g^p$is the corresponding (imputed) allocation of work. Note that $g_{max}^p = \max_i \{g_i^p\}$ would be the time between completed items if the line were operated as a classical assembly line based on the imputed allocation of work. Accordingly, g_{max}^p can be interpreted as the *imputed cycle time* of the line with workers in positions x^p. This interpretation will be useful to us in evaluating the productivity of TSS lines.

4.3 TSS As a Pull System

In our model a TSS system has complicated piecewise-linear dynamics that depend on the partition of work among the stations and the initial positions and velocities of the workers. However, we can show that, when the line is configured as a "pull" system by sequencing the workers from slowest to fastest, then the dynamics become dramatically simpler and very useful. In fact, the line becomes *self-organizing:* the workers will spontaneously and without intention space themselves so that the line produces at a constant rate. Moreover, if no station has "too much" work, then the production rate is P, the maximum achievable under any organization of the workers.

Let g^0 be the allocation of work corresponding to worker positions x^0. Then the orbit $\{g_{max}^p\}_{p=0}^{\infty}$ of any initial vector of gaps converges to a unique allocation

Theorem 1 *Let all workers be of distinct skill levels and sequenced from slowest to fastest* ($v_1 < \ldots . < v_n$,,); *then there exists a unique g* to which the successive allocations of work converge, independently of the initial allocation of work.*

Proof For convenience of the reader, we present the proof for a special case of our model in which workers are allowed to share stations (but not to pass each other). Alternatively, this may be seen as the limiting case of a process in which the number of stations is greatly increased so that each is assigned a tiny portion of the work content. In this simplified model, when the workers are sequenced from slowest to fastest, then no worker is ever blocked. This makes the analysis considerably simpler, yet includes all of the main ideas from the more complicated model.

Because no worker is ever blocked, the imputed allocation of work of a line changes after each phase as follows: $g_i^{p+1} = g^p$, and, for $i = 2, \ldots, n$,

$$g_i^{p+1} = ((x_i^p + v_i g_n^p) - (x_{i-1}^p + v_{i-1} g_n^p))/v_i \quad (2)$$

$$= (x_i^p/v_i + (x_{i-1}^p + (1 - v_{i-1}/v_i) g_n^p \quad (3)$$

$$= (v_{i-1}/v_i) g_{i-1}^p + (1 - v_{i-1}/v_i) g_n^{p+1}, \quad (4)$$

where expression 4 follows from expression 1, the definition of gap. We can write these equations as a linear system

$$\mathbf{g}^{p+1} = \mathbf{T}\mathbf{g}^p,$$

where each row of the matrix T sums to 1 and, because $v_{i-1} < v_i$, each element T_{ij} is in [0,1]. Therefore T can be viewed as the transition matrix of a Markov chain that, it is straightforward to show, is irreducible and ergodic. Thus, by the fundamental theorem of Markov Chains, there exists a unique $\mathbf{g}^*$ such that, for any $\mathbf{g}^0$, $\lim_{k \to \infty} T^k \mathbf{g}^0 = \mathbf{g}^*$ (see Ross, 1980, for example).

□

Unfortunately, if at this stable configuration some workers are blocked, then the maximum production rate might not be achieved. However, when no workers are blocked at the stable configuration, then the production rate is P, the largest that can be achieved by *any* way of organizing the given workers, even if one were to provide them with additional work stations.

Corollary 1 If *the system is at a fixed point at which each worker finishes at his initial station before the preceding worker attempts to use it, then*

1. Worker i repeatedly executes the same interval of work content

$$\left[\frac{\sum_{j=1}^{i-1} v_j}{P}, \frac{\sum_{j=1}^{i} v_j}{P} \right]$$

2. All workers invest the same work, as measured by the clock, in each item produced.

3. The production rate is P, the largest possible.

Proof We find the value of $\mathbf{g}^*$ by solving the system of equations $\mathbf{Tg}^* = \mathbf{g}^*$, and then use the definition of gap to solve for $\mathbf{x}^*$. The other claims follow by simple algebra. □

Theorem 1 and its corollary may be interpreted as showing that, to configure a bucket brigade from well to fire, one should put the fastest people close to the fire; then the people will, without intention, space themselves to convey the greatest possible flow of water upon the fire.

Note that the hypotheses of the corollary are easy to check: Worker i will complete his imputed allocation of work without being blocked if worker i+1 finishes at his initial station before worker i reaches it; that is, if

$$(e(x_{i+1}) - x_{i+1})/v_{i+1} \leq (b(x_{i+1}) - x_{i..+1})/v_{..i+1},$$

where $x_i = \frac{\sum_{j=1}^{i-1} v_j}{P}$ and $x_{i+1} = \frac{\sum_{j=1}^{i} v_j}{P}$.

Figures 16, 17, and 18 show the convergence of a system from three complementary points of view. Figure 16 shows an example of how the movement of the workers stabilizes, with the faster workers allocated more work; Figure 17 shows the convergence of the system within the state space of worker positions; and Figure 18 shows the average production rate converging to optimum. These simulations were generated by three workers of velocities v = (1, 2,3).

Our theorem suggests that, when configured as a pull system, the TSS line is robust in several senses. First, it will rebalance itself after a one-time disruption; for example, when a worker takes a break, the work content will be spontaneously reallocated among the remaining workers. It will also continually rebalance itself in the presence of noise, such as small variance in the time to complete a task. Furthermore, the line will rebalance itself to account for "drift," such as when workers tire and slow (as long as the workers remain sequenced from slowest to fastest). Finally, the line is self-balancing without knowing the statistics of task times or even worker velocities; all that is required is to know the *relative* velocities of the workers (who is faster).

Not only is the TSS line robust, so is our model: If an actual production line is close to our idealized model, then observed behavior will be close to predicted behavior. For example, if, contrary to our assumption, the time to walk back and preempt a worker is non-zero but small, then the TSS line will exhibit behavior close to that predicted by our model. Similarly, if one

worker slows slightly so that he is no longer faster than the person behind him, then–as long as the discrepancy is not great–the system remains close to optimum. Thus it might not matter that details of our model are simplistic, as long as they are not egregiously false.

4.4 Complicated Behavior

Even when workers are sequenced other than form slowest to fastest, there is always a stable allocation of work–however, the stable allocation might not be an attractor and so from an arbitrary starting position the allocation of work might not stabilize. Instead, the system can be trapped in highly complicated behavior patterns that are suboptimally productive.

Let f be the function, given implicitly by the TSS Rule, that maps $\mathbf{x}^p$ to $\mathbf{x}^{p+1}$

Lemma 1 *The function f is continuous.*

Proof Let x_i^p and y_i^p be two nearby alternative positions for worker i at the start of phase p. Then by the logic of the TSS Rule, at the end of the phase, the alternative positions of worker i can be no farther apart, so that after the line resets $|x_{i+1}^{p+1} - y_{i+1}^{p+1}| \le |x_i^p - y_i^p|$. Therefore, x^{p+1} is close to y^{p+1} whenever x^p is close to y^p. □

Theorem 2. For any given sequence of the workers on the line, there exists a fixed point x* such that if the workers start at positions x*, then (in the absence of perturbations) they will reset to x* at the end of each phase.

Proof. We continuously extend f so that its domain is the entire closed n-cell defined by $0 \le x_1 \le \le x_2 \le \ldots \le x_n \le 1$. To do this, make the natural extension to the TSS rule so that, if more than one worker should be assigned to the same station, then the worker with the item closest to completion (that is, greatest x_i) has priority and the others must wait to use this station until he has finished. (This extension is never necessary in practice; it is just a means of satisfying the hypotheses of the theorem we ill invoke.) Now, since f is continuous on an n-cell. by Brower's Fixed Point Theorem f has a fixed point (Bollobas, 1990). Furthermore, this fixed point must be within the natural domain of f because, by the logic of the extended TSS rule a point in the extended domain cannot remain fixed under f.

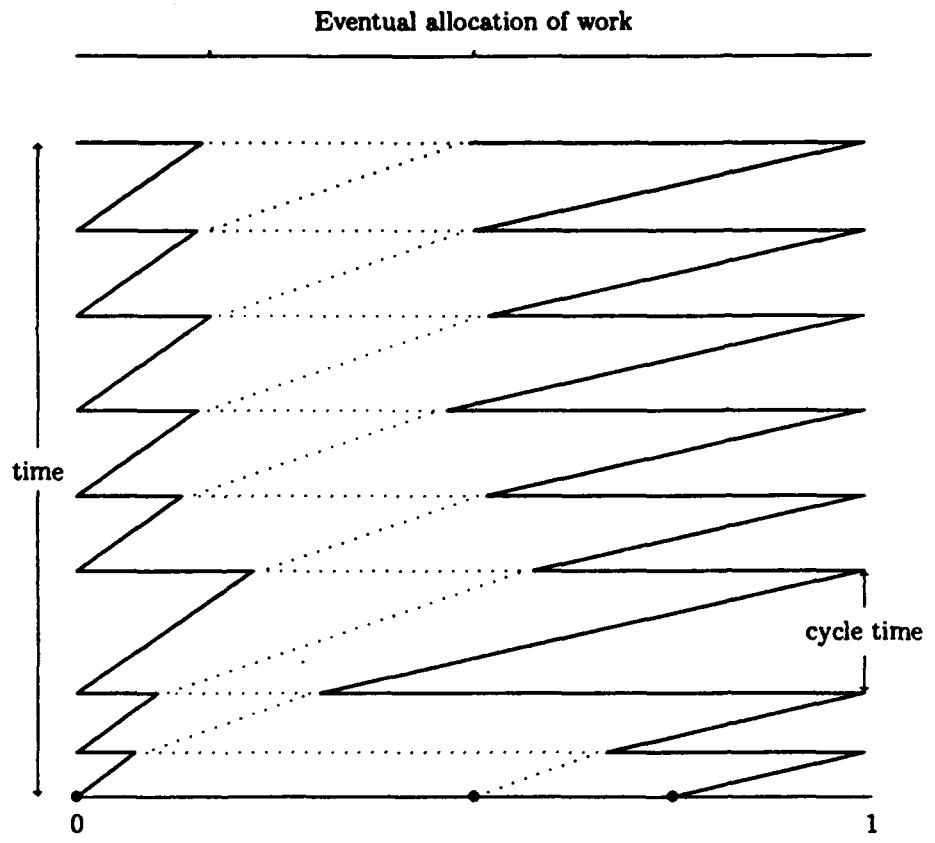


Figure 16 A time-expanded view of a TSS production line with workers sequenced from slowest to fastest. The solid horizontal line represents the total work content of the product and the solid circles represent the initial positions of the workers. The zigzag vertical lines show how these positions change over time and the rightmost spikes correspond to completed items. In this simulation the system quickly stabilized so that each worker repeatedly executes the same (optimal) portion of work content of the product.

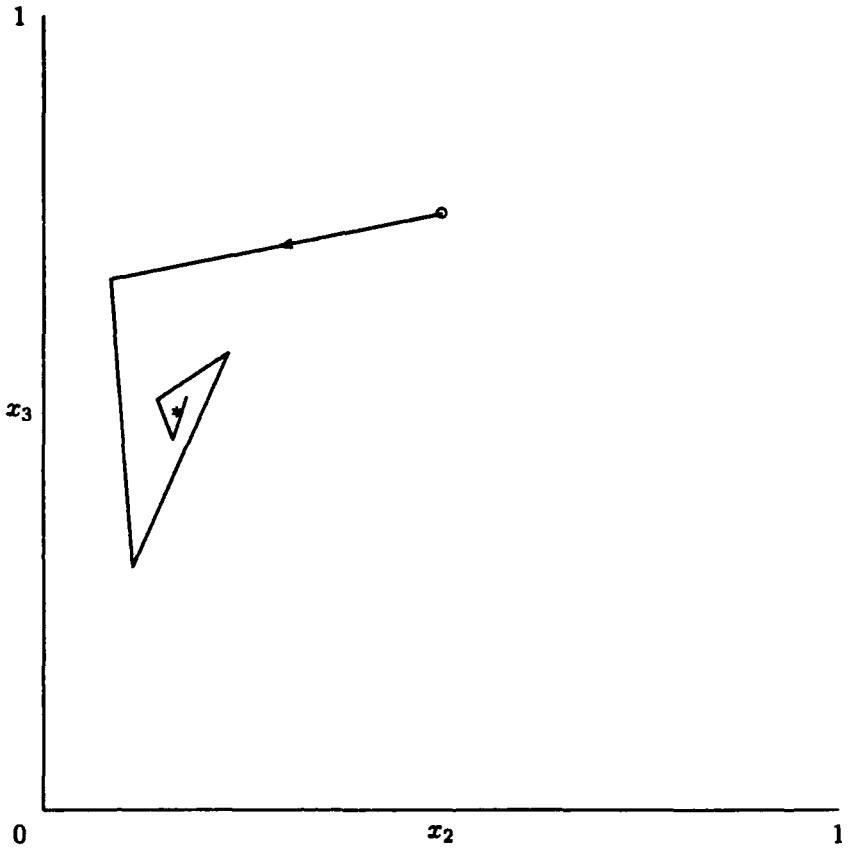


Figure 17 The positions of the workers on the production line at successive instants when it resets. (Since the position of the first worker is always 0 when the line resets, only (x_2, x_3), the positions of the second and third workers, are plotted here.) From any initial position the system converges. Here the fixed point is $(1/6, 1/2)$.

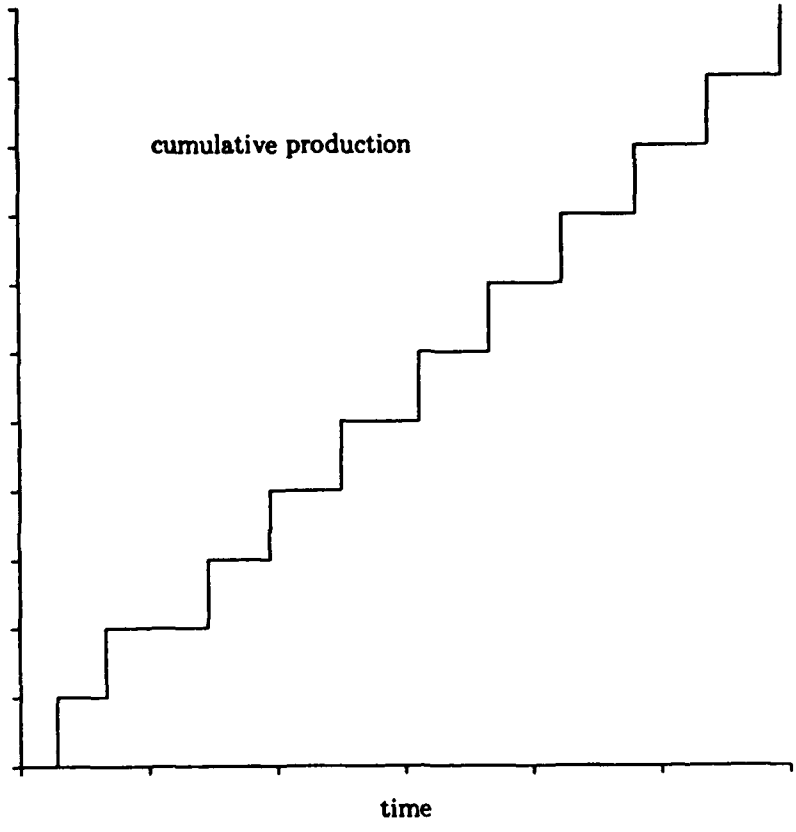


Figure18 When workers are sequenced from slowest to fastest, the cycle time converges to a unique value, independently of where the workers start on the line. In this instance, as the system approaches its limiting configuration, no worker is ever blocked; therefore the production rate approaches $P = 6$, the maximum possible.

Unfortunately, when workers are sequenced other that slowest to fastest, a fixed point can be unsustainable in the real world. The difficulty is that, when workers are sequenced other than from slowest to fastest, a fixed point can fail to be an attractor; moreover, the fixed point can, perversely, be a *repeller*, so that if the system ever deviates, however slightly, from the fixed point, then is must inexorably diverge from it (Devaney, 1989). In the real world, this deviation *must* occur because the data that determine the fixed point are not knowable exactly, and moreover, might change over time.

Figures 19 and 20 show a simulation in which the fixed point is a repeller and any orbit that strays must eventually be trapped by a limit cycle with production rate less than that of the fixed point. (This simulation was generated by three workers of velocities v = (3,2,1).) The suboptimality results from the fact that a faster worker can be repeatedly blocked by a slower worker, with consequent loss of production rate.

In other simulations we found instances of quite large cycles, some at the limits of the numerical resolution of our computer and the patience of the observers. For example, when v = (1.0,1.02,1.0) there is an attracting limit cycle of length 1159.

In simulations of a TSS line we also observed such phenomena as multiple fixed points of differing production rates (for a fixed set and sequence of workers) and limit cycles that depended on starting positions of the workers. Furthermore, even slight changes in the data (distribution of work content over the stations, initial positions of the workers, and, especially, values of the v_i) could result in wildly varying behavior of the line.

In practice, repeated, complicated reallocation of work is not necessarily a problem if each worker continues to visit the same subset of stations. Then it is possible that the only effect is, for example, that a worker might preempt the sewing of a long seam at different spots in successive resets. However, if the work allocations vary so that workers visit a changing set of stations, then the line can become difficult to manage. For example, in simulations some workers visited most of the stations while other colleagues were blocked by slower workers and visited few stations.

To enforce manageability, some commercial TSS lines are set up with a metarule that restricts each worker to a predetermined contiguous subset of stations. This has some (at least theoretical) disadvantages: It can waste productivity by forcing a worker to be idle; it reduces the flexibility of the line by making it more involved to add or remove workers; and it might not be necessary if workers are sequenced from slowest to fastest.

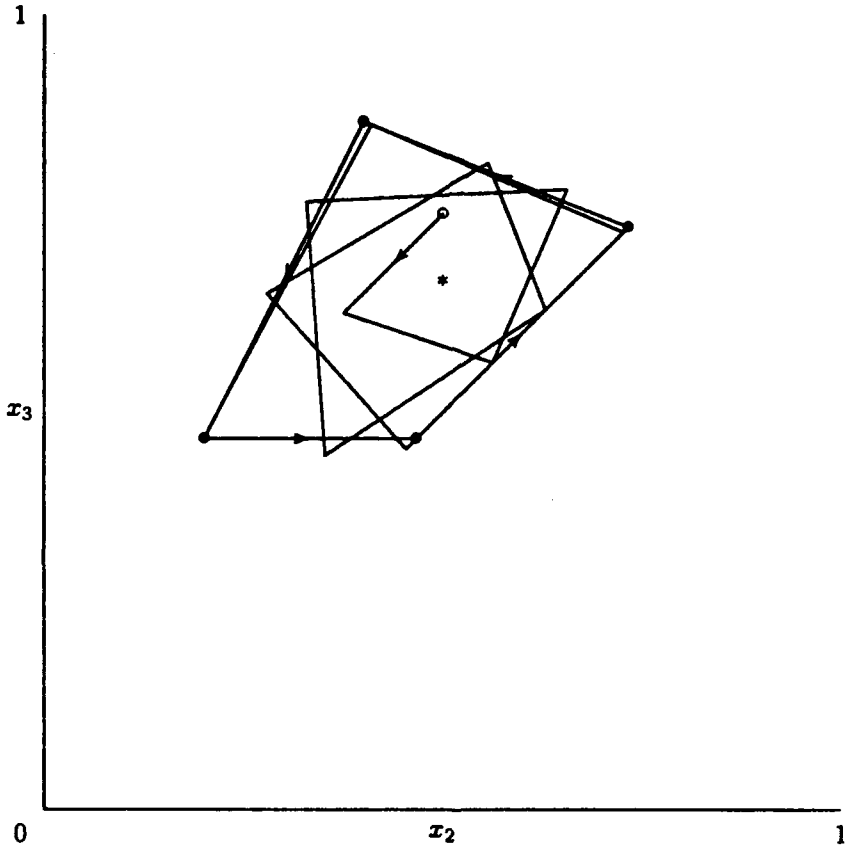


Figure 19 The positions of the workers on the production line at successive instants when it resets. In this instance the fixed point $(1/2, 2/3)$, indicated by $*$, is optimal, but a repeller; and any orbit that strays from it will be trapped by the attracting but suboptimal limit cycle consisting of the points $(3/15, 7/15)$, $(7/15, 7/15)$, $(11/15, 11/15)$, and $(2/5, 13/15)$, indicated by $\bullet$'s.

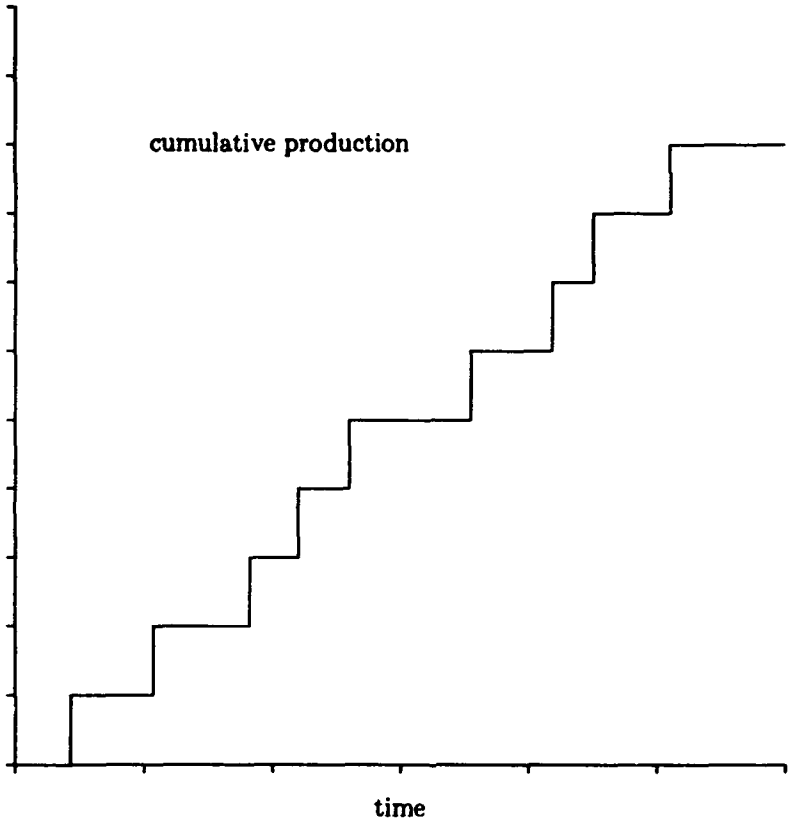


Figure 20 As the system is trapped by a limit cycle, the cycle time oscillates and the average production rate converges to 60/11, which is less than the maximum value of $P = 6$.

More troubling than complicated behavior is anomalous behavior: When workers are sequenced other than from slowest to fastest, it can happen that the production rate can be *increased* by slowing or even by removing a worker. For example consider the TSS line with **p** = (1/2,1/4,1/4) and **v** = (2,1, 2). The production rate of this line increases if worker 3 slows to v_3 = 1. The production rate will also increase if worker 2 is removed from the line. (The converse behavior holds as well, in which adding a worker or speeding up a worker causes the production rate to fall.)

We emphasize that none of this complicated or anomalous behavior pertains when workers are sequenced from slowest to fastest. Using the above arguments we can state the following theorem.

Theorem 3 *When a TSS line is configured as a pull system, the production rate can never be increased by slowing or removing a worker.*

Incidentally, we have been told that commercial TSS lines are frequently configured with the fastest workers at the very first and very last positions. If this actually increases the production rate, we conjecture that it is due to psychological factors. Our model predicts no benefits from putting a fast worker first on the line.

4.5 Related Work

There has been a wealth of work on production lines, far too much to survey here, but most differs from ours in not modeling the worker explicitly. Generally the worker is simply identifies with a work station the location of which remains fixed. Our model seems unusual in that we treat workers as resources distinct from the stations at which they might work; moreover, we model even individual workers by specifying their skill levels.

Ostolaza, McClain, and Thomas (1991) described assembly lines with some additional flexibility: For each pair of adjacent work stations j and j+1 there is a shared task p_j that may be done at either station; furthermore, that decision can be made during production. Ostolaza et al., gave evidence that, if all processing times are exponentially distributed, then the line can function effectively even if each station uses a simple rule to decide which of its waiting tasks to perform next and whether to perform the shared task itself or to pass responsibility for it to the following station. Their model was intended to be suggestive, since the distribution of task times is chosen for tractability rather than plausibility.

The production line described by Ostolaza et al., differs in several ways from the TSS line. First, workers are identified with work stations and the line is balanced by clever management of work-in-process inventory. In contrast, the workers on a TSS line move to where the work is and so there is no work-in-process inventory beyond that in the hands of the workers. Another difference is that the line of Ostolaza et al., does not allow tasks to be split between work stations (workers), while the TSS line allows task preemption at any time. This suggests that the line of Ostolaza et al., might be more appropriate where task preemption is very costly and it is not too expensive to provide two sets of tools for each of the shared tasks. A TSS line seems more appropriate when preemption is not costly and tools are expensive, as in the apparel industry.

Even if preemption is costly, one can amortize preemption costs under TSS by manufacturing batches rather than single items. By the logic of the TSS Rule there can be at most n preemptions during the manufacture of a batch. Manufacturing batches of size k reduces the per-unit preemption cost by a factor of k. Of course the savings in preemption costs must be balanced against the cost of a k-fold increase in work-in-process inventory.

Schroer, et al., built a simulation model of a particular TSS line (Schroer, Wang, and Ziemke, 1991). However, the point of their work was to demonstrate capabilities in object-oriented simulation and so they did not pursue analysis of the TSS system. Instead they were content to gather statistics and observe that the line seemed to work well. Our model predicts analytically all of the statistics they gathered through simulation.

4.6 Implementation of the TSS rules at the AMTC Demonstration Center

The TSS Rules were implemented at the AMTC demonstration center at Southern Tech in the final phase of this project. In the first two experiments we wanted to observe how the allocation of work changed with different allocations of workers to positions in the line. The third experiment demonstrates how the production rate can depend on the number of units handled at the same time by each worker. Seven operations were performed in the module for assembly of milspec BDUs. Each worker started from the first operation on their first garment, then operated under the TSS rules. The results of the experiments are presented below.

Experiment #1 (two workers on the line)

1-A: Workers arranged ___>_____>
slowest fastest

Timings taken for each BDU were at the hand-off from the first to the second worker and at the time the garment was finished. The two operators were able to complete 6 garments in 30 minutes.

BDU number	Hand-off Time*	Finish Time*
1	---	10.55
2	8.28	11.46
3	2.997	8.333
4	6.02	9.58
5	3.856	9.14
6	5.18	9.14

* Times are in minutes.

Figure 21. Work allocation for 2 workers arranged slowest to fastest.

1-B: Workers arranged ___>_____>
fastest slowest

All other factors are the same as 1-A.

BDU number	Hand-off Time	Finish Time
1	---	10.37
2	8.88	11.28
3	2.17	14.031
4	11.84	13.49
5	2.075	12.575
6	10.29	12.57

Figure 22. Work allocation for 2 workers arranged fastest to slowest.

Observe that completion of six garments required considerably longer than 30 minutes in this case. The problems arising were to be expected since the slowest worker was at the front of the line. The fast worker spent a lot of time waiting to hand-off her garment before returning to the beginning of the line to introduce a new unit. This is attributed not only to the inappropriate arrangement of the workers, but also to the lack of experience of the slow worker on the operations at the end of the line.

Experiment #2 (three workers on the line)

This experiment is exactly like #1, with the addition of a third worker.

2-A: Workers arranged ___>_____>
slowest fastest

BDU number	First Hand-off Time	Second Hand-off Time	Finish Time*
1	---	---	12.22
2	---	11.02	14.72
3	8.27	3.68	14.17
4	3.76	2.14	14.03
5	2.15	8.09	12.92
6	8.03	2.67	13.00
7	2.58	2.37	12.39

Figure 23. Work allocation for 3 workers arranged slowest to fastest.

2-B: Workers arranged ___>_____>
fastest slowest

BDU number	First Hand-off Time	Second Hand-off Time	Finish Time*
1	---	---	.9.99
2	---	.8.21	12.04
3	.6.03	4.16	11.50
4	3.99	1.61	9.99
5	1.70	4.27	10.22
6	4.29	4.27	12.25
7	4.26	3.64	10.35

Figure 24. Work allocation for 3 workers arranged fastest to slowest.

Experiment #3

In all of these runs, three workers arranged ___>_____>.
slowest fastest

The workers all begin at station 1 and introduce one, two, or three garments (in that order) in each of the succeeding runs. This group of garments is moved along the line as a "unit."

3A. The unit is one (1) garment. Total production was 15.889 garments.
Max WIP = 3. 15 completed garments, 1 was 59.5% complete, and 1 was 29.4% complete.

3B. The unit is two (2) garments. Total production was 15.945 garments.
Max WIP = 6. 14 completed garments, 2 @ 59.5%, 1 @ 46.1% and 1 @ 29.4%.

3C. The unit is three (3) garments. Total production was 15.52 garments.
Max WIP = 9. 12 completed garments, 3 @ 71.6%, 2 @ 46.15%, 1 @ 29.4%, and 1 @ 15.6%.

There are two main points we see from these experiments that extend the current practice of TSS. (1) It is apparently better to sequence workers from slowest to fastest. You get spontaneous plus higher production rate. Interestingly, you can interpret this as placing the bottleneck to production **first** on the line. This appears to contradict the traditional wisdom that one should put the fastest worker first because "you can't get it out until you get it in;" but in fact, since TSS is a "pull" system, you can't get it in until you get it out. (2) Despite claims by Americas 21st, the size of a "unit" is an issue. They say a unit is one item. One might actually get better production throughput by defining a unit to be more than one item. Of course, it is possible to push this too far; if a unit is large enough then you are back to the bundle system!

4.7 Conclusions

We have examined several TSS lines in operation, courtesy of Americas 21st. A typical one, which produced trousers, was configured with three workers and seven stations. The stations were arranged in a "U"-shape to reduce travel time of the workers. The total time for the line to reset was 7-10 seconds, which was an order of magnitude less than the work at a station (45-90 seconds) and two orders of magnitude less than the total work content of the product (about 6 minutes). In light of this, it seems reasonable that our model assume instantaneous resets.

The TSS line has many attractive properties. First, it is *effective* in that it can achieve maximum production rate. The TSS Rule is *simple*, which makes it easy for the workers to learn.

It is *parsimonious* in its data requirements, which are only the relative speeds of the workers (not even their values); and it does not require knowledge of task times. It is *adaptive:* The line configures itself without management intervention. Finally, the TSS line has negligible work-in-process inventory: one item for each worker.

A TSS line can also be implemented "on top of" a classical assembly line: First the tasks are distributed over the stations, which will typically result in an imperfect allocation of work among stations. This first allocation is static and unchanging. Then workers, sequenced from slowest to fastest, follow the TSS rule on the line. A second allocation of work emerges, this time among the workers. This second allocation is dynamic and self-correcting and it can smooth over imperfections in the underlying static allocation.

There seem to be two main issues in configuring a TSS line. First, the workers should be sequenced from slowest to fastest so that the line will be selfbalancing–then the allocation of work will stabilize. Next, bottlenecks should be eliminated so that, after the line has balanced itself, no workers are blocked– then the production rate will be P, the largest possible for *any* way of organizing the given set of workers.

Stations early in the line are more likely to be bottlenecks to production because those stations are staffed by the slower workers. The simplest way of avoiding bottlenecks is to move work or stations with high work content toward the end of the line (insofar as allowed by precedence constraints among the tasks). Toward the end of the line such stations will be staffed by faster workers and so require less clock time. Alternatively, one could resort to standard fixes such as replacing a bottleneck station with two new stations that partition the work of the original.

Incidentally, in studying simulations of TSS we noticed some interesting behavior when there were multiple bottleneck stations on a line on which workers had been sequenced from slowest to fastest. If we eliminated the last bottleneck, then faster workers that would have otherwise been blocked at this station were reallocated toward the front of the line. The earlier stations were no longer bottlenecks when staffed by the faster workers. Thus reducing the work at the last bottleneck eliminated all preceding bottlenecks. In our experiments the converse did not hold.

At first glance it appears that a TSS line can be improved, or at least equaled by changing the implementation slightly. For example, it seems natural to consider having the workers circle through the work stations, so that after a worker finishes an item, he begins a new one at the

work station. This avoids preemption altogether, but it requires that every worker be trained at every task. Worse, such a line can sustain a production rate no greater than that of the slowest worker, because all the others will eventually catch up to him and must queue behind him. This cannot be fixed by allowing workers to preempt in the forward direction because a preempted worker would have to remain idle until his work station became free again, and so the line could not keep all workers busy.

Another variation of TSS that seems appealing at first glance but fails to work as well is this: Each worker, when blocked, leaves his partially completed item in a buffer before the busy station and then follows the backward part of the TSS Rule. Unfortunately, under this variation workers tend to migrate to the region preceding a bottleneck station, where work-in-process inventory accumulates. Furthermore, the resulting allocation of work is quite unbalanced and the production rate suffers.

Finally, we mention that the following simplified model can be useful in estimating the production rate or the appropriate number of workers for a TSS line. When all workers are of almost the same skill level, then since the work standard can be rescaled, we can assume without loss of generality that all $v_i \approx 1$. In this case the maximum production rate of n workers is $\min\{n, 1/p_{\max}\}$ items per time unit; and it is straightforward to show that the TSS line achieves this rate on average over each n consecutive items (after the first n). This expression for production rate makes clear the effect of adding additional workers: both production rate and station utilization increase proportionally with n until $n \geq 1/p_{\max}$, at which point they increase no further.

5.0 The Virtual Manufacturing Enterprise

5.1 Introduction

A manufacturing enterprise is a system composed of components exhibiting significant decentralized decision making and control. A Virtual Manufacturing Enterprise (VME) is a network of functional modules implemented in software, where each module emulates some component of the manufacturing enterprise. Using this concept, it is not incorrect to say that each of the modules is a VME in itself. In this work we have constructed one such module for the apparel manufacturing enterprise, a VME to model the flexible work group. This model is very flexible and can be used to emulate the real time operation of a work group on the shop floor.

Conventional wisdom says that manufacturing enterprises can only be studied in the field. Such studies are very expensive and, as a result, often very limited. To determine a

change to the system, the actual system must be changed. Since components cannot be isolated and studied in a laboratory environment, it is very difficult to understand the interactions among components. Also, it is impossible to field study conceptual systems which have not yet been built. A major difficulty with studying a manufacturing enterprise in a laboratory is the methodology for adequately emulating a field environment has not been developed.

This virtual manufacturing enterprise will provide more than the capability to simulate the actual factory; the goal is to emulate the system. With each functional component modeled in software, and each of these operating independently of the others, it will be possible to dynamically interact with one or more processes which the remainder of the factory continues to operate as normal. In a real industrial setting, this is how such an interruption would actually transpire. If the automatic belt loop maker experienced a breakdown, it does not mean the entire sewing room must shut down in order to deal with that problem. So will the VME operate.

The ability to study alternative configurations and design concepts for the FWG, and the ability for real-time tracking of the system operation will be inherent to the design of the facility. During the course of this project, the system developed models only a the emulation of modular manufacturing. However, the system architecture will be designed and implemented so that later expansion to a full-fledged factory emulator can be easily accomplished.

Our VME has such a large scale architecture. The system has the following abilities.

A. runs in either real time or simulated time, i.e., time must be scalable.

B. runs on a network, with pieces of the VME distributed throughout the network, and at the same time all components being synchronized.

C. is independent of whether the pieces are real or software emulation.

D. allows different pieces to operate on different time scales (to allow studying some part more carefully than others).

E. is recursive in design so that each piece can be composed of still more pieces (again allowing us to model in more detail where needed).

In section 5.2 a detailed architectural description of the Flexible Work Group VME will be given. Section 5.3 describes how to operate the VME.

5.2 The VME Architecture

5.2.1 Description of Classes

In this section we describe the set of the classes in **vmeclass.c**. Literal names of functions, variables, classes, etc., are boldfaced.

class slist

This is a simple class to implement a (one-way) linked list of **void** pointers. The idea is to re-derive the class for whatever types you need, casting the **void** pointer to the appropriate type. Operations include appending to the list, stepping to next node in the list, getting value of the **void** pointer at a node, and deleting a node.

class node

This class implements a doubly linked circular list. Each node on the list has a name (**char** array) and a type (enumerated type **object_types**) associated with it. Operations include creating a node, adding or changing the type and name fields of a node, and searching the list for a node with a certain name. The purpose of **node** is to act as a parent class for the simulation objects and allow garbage collection; every object in the simulation is on the same **node** list.

class hash_list

This class is derived from **slist**, thus it is a singly linked list. The **void** pointer in **slist** is cast to a **hash_node** pointer. **hash_node** is a structure defined to have a **char** pointer and a **void** pointer. The **char** pointer points at an exact string, used to index an entry in the list, and the **void** pointer points at some information structure. **hash_list**, along with the class **hash**, is meant to be re-derived for whatever kind of hash table is needed.

class hash

Implements an array of **hash_lists** in order to support a full hash table. One routine, **new_entry**, adds a new entry consisting of an identification string and a pointer. **new_entry** hashes the string to get an index into the array of **hash_lists**. Then the string and pointer, stored in a **hash_node** structure, are added to the end of the **hash_list**. The routine **find** reverses this operation; given a string, it will find the correct **hash_node** structure and return the corresponding information pointer.

class buffer_table

Derived from the **hash** class, this implements a hash table of **buffer** pointers (see class **buffer**).

class cell_table

Also derived from the **hash** class, this is a hash table of **cell** pointers (see class **cell**).

class eval_elem

This class is used in executing the simulation. Also, it is derived from **slist**; it supports a singly linked list of **cell** pointers. The order of this list corresponds to the order in which the simulation objects will be evaluated during a clock tick. Because of the discrete nature of time in a computer simulation, a true parallel simulation is difficult to achieve. Instead, the simulation is evaluated one object at a time, in reverse order of the process flow.

class executable

This class is derived from **eval_elem**, **cell_table**, and **buffer_table**. The constructor builds an **eval_elem** list, analogous to compiling a program. The function **iterate** evaluates one step (clock tick) in the simulation.

class simulation_clock

Keeps track of the time in the simulation. Has some hooks to support real time simulations, but effectively this is just a counter.

class screen_image

This class holds and manipulates information about the image (**Widget**) corresponding to a given simulation object, such as a **buffer** or a **cell**.

class buffer

This class is used to specify connections between the **cells**. Objects awaiting processing wait in an input buffer, and are put into an output buffer after being worked on. There is one **buffer** between each pair of connected **cells** (one cell's output buffer is the next cell's input buffer). A **buffer** has fields describing which **cells** it is connected to, what kind of items it holds (the name of the items is in a **char** array), and how many items it holds (a **long**). On the screen, a buffer is represented by a **Widget** that looks like a thermometer.

class buffer_list

Derived from **slist**, this class implements a list of buffers. The idea is that a cell may have several inputs and outputs, and a variable length list of **buffer** pointers is more robust than a fixed array of input and output **buffer** pointers.

class cell

A **cell** is the basic unit of the simulation. It roughly corresponds to a (real life) machine or process. It maintains a list of input and output buffer. **cell** is not meant to be used in this form, but to be a base class for machines with specific properties. During a time tick in the simulation, a **cell** decides whether it is done with its current operation or not. If so, it puts its contents into the appropriate output buffers and gets more items from its input buffers.

class sequential_machine

Derived from **cell**, this is the simplest notion of a machine. It takes in one kind of thing from its input buffers, 'processes' the items for some number of time ticks, and puts the result into its output buffers. It has fields to keep track of: how long it takes to do a process, how many input items are needed, and how many items to put out when done. A real life example of a sequential machine is a fabric cutter: one kind of input, fabric, and one kind of output, cut fabric. The **Widget** for a **sequential_machine** is a sewing machine.

class spec_list

Derived from **slist**, this implements a list of **spec_node** structures. Each **spec_node** structure has text field and a **buffer_list**. This class is meant to be used with the heterogeneous machine type, where several different types of input are allowed. Each **spec_node** structure will correspond to a kind of input.

class hetero_machine

This class is more complicated than the **sequential_machine**, rather than taking only one kind of input, a **hetero_machine** can take several. A real life example would be a machine that assembles skateboards. The input parts (wheels, trucks, deck) are different. At the end of the operation, one kind of item comes out, an assembled skateboard.

class source

This is a derivative of **sequential_machine** that has no input. **source** is meant to start a simulation (i.e., be the first **cell** in the simulation). The **Widget** for **source** is an arrow coming up out of a hole.

class sink

This is also a derivative of **sequential_machine**, and has no output. **sink** is meant to be the last **cell** in a simulation. The **pixmap** for **sink** is an arrow pointing into a hole.

class parallel_machine

This class is meant to model a passive machine or process where the number of items operated on is large. A real life example would be chairs that have been painted. Any number of chairs can be drying at the same time, and new chairs can be added at any time.

class process

This is a derivative of **cells** that supports both inputs and outputs. It contains a sub-level of the simulation that the user can descend into. Once in the sub-level, users can create and connect additional machines that will all have the process as their parent. Processes can themselves contain processes and can be used to represent groups, departments, manufacturing processes and generally any operation that requires more than one machine.

5.2.2 Description of Source Files

constant.h Include file with **#defines** for **Widgets**, i.e., each **#define** indexes a global array of widgets. In the SUN version, this has been somewhat replaced by **vme_widg.h**.

structs.h Include file defining some structures for **vme_chan.c**.

vmeerror.c Implements a global error handler. Appends an error message to a file hardcoded at the beginning of this module (currently, this file is "vmeerror.log"). The format of the error call is:

error_handler(error_type level, **char*** format, ...)

where level can be **MILD**, **SERIOUS**, or **FATAL** (specified in the **vmeconsts.h** file). The other arguments are a **printf** style format string and a variable number of arguments. An example of a call to the error handler is:

error_handler(SERIOUS, "bad argument: bee(%d)\n", i);

vmeslist.c Methods for the **slist** class.

vme_buff.c Methods for the **buffer** class.

vme_cell.c Methods for class **cell**.

vme_chan.c More **X Windows** support code.

vme_copy.c Functions to make a copy of a cell. These routines are used by the edit code; when a cell is edited, a temporary copy is made so as to allow the user to cancel the edit. When the 'apply' button is clicked, the temporary cell is copied back to the original.

vme_ctrl.c Supplies functions to hook in the "control software," i.e., user defined functions.

vme_eval.c This is an outdated module; at the present time it is empty.

vme_exec.c Methods for class **executable**. This class is used to 'compile' the simulation: when the constructor is invoked, it constructs a linear list of cells, corresponding to the order in which they should be evaluated during a time tick. (This is necessary because the simulation engine does not support parallel operations -- there is a definite ordering). The list is constructed based on a breadth first search of the cells, with buffers connecting cells, starting with the last cell in the process (a **sink** cell).

vme_hash.c Methods for the class **hash_list**.

vme_hete.c Methods for the class **hetero_machine**.

vme_htab.c Methods for class **hash**. Includes routines for hashing a string, searching the hash list, and adding an entry to the hash list. Relies on routines defined in the module **vme_hash.c**.

vme_load.c Loads a simulation from a file. Recognizes files generated by **vme_save.c** as well as free format user files. Contains a relatively sophisticated parser.

vme_node.c Methods for class **node**, a circular doubly linked list. There should be one node ring, corresponding to a simulation. Each **node** corresponds to a simulation object. The ordering of the list has nothing to do with the simulation; this list is to allow global searches of cells and buffers, and to facilitate garbage collection.

vme_para.c Contains methods for class **parallel_machine**.

vme_save.c Saves a simulation to a file by generating definitions for each object and connection in the simulation. Each simulation object has a function called **generate_cell_definition** (if the object is a cell) or **generate_buffer_definition**. The string returned by each of these calls is written to a file.

vme_scrn.c Associated with the class **screen_image**, this module acts as an interface between the simulation engine and the **X Windows Widgets** that you see on the screen.

vme_seqm.c Defines the methods for the class **sequential_machine**.

vme_sink.c Methods for the **sink** class.

vme_sour.c Methods for the **source** class. This is derived from the sequential machine class, so it is a small module.

vme_spec.c Methods for the **spec_list** class, used by the heterogeneous machine class.

vme_widg.h See description of **constant.h**.

5.3 How to Operate the VME

5.3.1 A Sample Simulation

Example of a Simulation

This section describes the steps in creating and running a simulation via a simple example. This example assumes that the simulation is created interactively, while running the program, rather than by editing a **.sim** file (**.sim** is the file suffix for simulation files). Also, the descriptions of pulldown menus, etc. are not exact as there are some differences between the Vax and Sun implementations. The user should be familiar with graphical user interfaces, i.e., know how to open a menu or drag an item across the screen with the mouse.

The first step is to run the program; this is done by typing **vme** at the command line prompt (for the VAX version, type **run vme**). A large empty window will pop up, with a **GT** logo in the upper left hand corner and a row of menu buttons across the top of the window. The buttons will read "File Edit Run Display Customize", in that order. Moving the mouse arrow over one of these buttons and pressing the left mouse button will open that menu. From here on, moving the mouse arrow over an item and pressing the left mouse button will be referred to as **clicking** on that item. To select an item on a menu, click the menu button to open the menu, holding the mouse button down, and move the mouse arrow down until it is over the selection you want. The name of the selection will be highlighted; let go of the mouse button to select it. You can quit the program at any time by opening the **File** menu and selecting **Quit**.

We'll start out with a very simple simulation with just 3 elements: a source, a sink, and a sequential machine. Sources and sinks are used to define the beginning and the end of a simulation and usually don't have a physical analog. Though there are three elements in the simulation, only the sequential machine corresponds to a real life machine. In this case, the sequential machine will represent a sewing machine.

Since we are starting with an empty simulation, the first step is to create the needed cells. The source, sink and sequential machine mentioned previously are all examples of cells. Connections between cells are called buffers, and will always appear as a vertical bar graph in a box.

The **edit** menu allows the user to create, delete, and edit cells in the simulation. We will create the three cells by using the **create** entry of the **edit** menu three times. Go ahead and select it now; a window will pop up allowing the user to select the kind of cell and the name. Type "start" in the name field at the top of the new window; this is the name of the cell we are creating. Select a source cell from the **machine type** list with the mouse by moving th mouse arrow over the text that says source and pressing the left mouse button. The text will now be highlighted; click the **Okay** button to actually create the cell and close the window. Generally, wherever an **Okay** button appears, there is also a **Cancel** button. If you make a mistake or change your mind about something (and this is true for any of the popup windows in the simulation), click the **Cancel** button to tell the program to "forget about it." Now a start icon will appear in the upper left corner of the screen. Move the icon to the middle left of the screen by dragging it with the mouse: move the mouse arrow over the icon, press the left mouse button down and hold it (the arrow will turn into a small hand), and move the hand to where you want to put the icon. Let go of the left mouse button to complete the move. Now create a sequential machine cell and name it **sewing machine**, dragging it to the center of the screen after you have created it. Also create a sink cell and name it **end**, and drag that to the middle right of the screen.

The next step is to specify the connections between the cells. Move the mouse cursor so that it is over the middle machine and click the left mouse button once. The sequential machine icon will be highlighted. Select the **edit** entry of the **edit menu**. A large edit window will pop up. Click the **add** button below the **input from** list at the lower left corner of the window. The **new input** menu will pop up. Type the name of the connection in the name field -- call it **cloth**. Then use the mouse to select the entry on the list labeled **start** (the list is the cells available to connect to). When **start** is highlighted, click on the **Okay** button to close the window and implement the change. Click the mouse on the quantity field, below the word **Quantity**, next to

the **input** list. The default value is -1; change it to 10. There is no need to hit return; if you do, the simulation will make the change and close the window, just as the **Okay** button does. If you do inadvertently close the edit window, you can just open it again using the **edit** selection. Now click the **add** button below the **output to** list at the lower right corner of the edit window. A menu very similar to the **new input** menu will pop up, only this one is to specify the output connections. Type **shirts** for the name of the connection. Select **end** from the list and click the **Okay** button. Click the mouse on the quantity field next to the **output to** list and change the default setting to 10. At the top of the edit window, click on the **production rate**; change the default value to 5. Now click on the **Okay** button at the bottom of the edit window. We have specified the connections in the simulation, their capacities, and the rate at which the machine labeled **sewing machine** works.

Now we need to specify the production rates for the other two machines in the simulation **start** and **end**. Click on the **start** icon to highlight it, and select **edit** from the **edit** menu. An edit menu will pop up. Notice that under the **output to** list, **sewing machine** is already specified. Click on the **production rate** and change the default value to 10. Now click **Okay** to make the change; the edit window will disappear. Click on the **end** icon to highlight it, and again select **edit** from the **edit** menu. When the edit menu pops up, **sewing machine** will be on the input list. Click on the **production rate** and change the default value to 10. Click the **Okay** button.

Now we are ready to run the simulation. But first, it is always a good idea to save your work. Click the **file** button and select the **save** entry. For the file name, type **simple.sim**. Click the **Okay** button or hit return to save the simulation and close the window. The file **simple.sim** will look like the following.

```
# simple.sim cell definitions

size_window(x_pos=600, y_pos=500)

# "World Cell" is virtual.

sink(thru_put=10,num_in=1,cell_name="end",x_pos=505,y_pos=205);

sequential_machine(thru_put=5,num_in=1,num_out=1,cell_name="sewing
machine",x_pos=266,y_pos=210);

source(thru_put=10,num_out=5,cell_name="start",x_pos=54,y_pos=215);
```

```
# Connections between cells

connect_output(from="sewing machine",to="end",what="shirt",capacity=10);

connect_output(from="start",to="sewing machine",what="cloth",capacity=10);
```

Now we have to compile the simulation by selecting the **reset** entry on the **run** menu. This is a necessary step before the first run during a session and after a cell has been added or deleted from the simulation. It is analogous to compiling a program: if a change has been made, the program needs to be recompiled. A status box will pop up when you select **reset**. Use the mouse to drag it off to the side of the main window. Now the simulation is ready to run: select the **start** entry from the **run** menu.

In running the simulation, the thing to watch is the buffers (they look like bar graphs or thermometers). They will be 0% full when the simulation is first started. As the simulation runs, and under our parameter values, we see the buffer between source and sequential machine slowly fill up. The sewing machine can't keep up with the supply.

Now stop the simulation by clicking the **stop** entry of the **run** menu. Edit the sequential machine cell (the middle one) and increase its work capacity by making the production field 10 rather than 5. We don't have to recompile the simulation because we have only edited a cell, not added or deleted one. Run the simulation again. This time the buffer between **start** and **sewing machine** stays near 0% full. Quit the simulation by selecting the **quit** menu.

vme_load.c and .sim files

Vme_load.c is the source file for the routine that takes care of loading in a simulation. Simulations are saved to files with a suffix of .sim. They can be created automatically by the simulation, created by using a text editor, or a combination of the two. The file format of a .sim file is purposefully made flexible to make it easy for the user to edit their own .sim files.

5.3.2 The Preprocessor

When a line of text from a .sim file is read in, several things happen. The line is truncated at the first # not in a quote (sandwiched between "'s). Anything after the # is taken to be a comment. All whitespace characters (spaces, tabs, etc.) are stripped, and all upper case characters are converted to lower case, except for quotes. The first occurrence of a " character signals the beginning of a quote. All characters after that are ignored until the next ", which

signals the end of the quote. The preprocessor will not change anything in quotes; it will pass it through untouched. An example would be, given the line:

Connect_Input (FROM="this", To = "THAT!"); # comment

The preprocessor would pass the following line to the parser:

connect_input(from="this",to="THAT!");

Also, if the parser gets an argument it cannot match, or an error in the transition table, it will throw away that line and try to restart on the next one.

5.3.3 Transition Table

The following transition table describes the pattern matching process that vme_load.c goes through when it reads a line of text. The leftmost column describes the current state of the parser. The top row describes the current token. The entry indexed by the state and the token is the next state of the parser.

	letter	_	(	)	=	number	"	,	other
Start	1		/	/	/	/	/	/	/
1	1		1	2	/	/	/	/	/
2	3		/	/	/	/	/	/	/
3	3		3	/	/	4	/	/	/
4	/		/	/	/	/	6	5a	/
5a	5a	5a	5a	5a	5a	5a	5b	5a	5a
5b	/		/	/	End	/	/	/	2
6	/		/	/	End	/	6	/	2
End	/		/	/	/	/	/	/	/

Start = wait until prefix is seen

1 = get prefix name, quit when a (is seen

2 = wait for argument

3 = kind of argument, quit on = sign

4 = decide whether quote or number

5a/5b = quote

6 = number

End = end of this line

/ = not an allowed transition

5.3.4 Format of an Input Line

A typical line in a .sim file is:

sequential_machine(thru_put=5, num_in=1, num_out=1, *cell_name= "sewing machine"*, x_pos=266, y_pos=210);

The first text, **sequential_machine**, describes what is being created. It is called a prefix. Given a prefix, the general form of a line is

prefix(arg1 = value, arg2 = value, ...)

Allowed prefixes are:

connect_output	connect_input
sequential_machine	sink
source	parallel_machine
hetero_machine	size_window
process_machine	

Connect_output and connect input create buffer connections between cells. Size_window specifies the size of the main window. If the new size is smaller than the default

size, the request is ignored. The other prefixes create cells of the same names (i.e., source creates a source cell). Allowed arguments are

from	to	what
cell_name	thru_put	num_in
num_out	x_pos	y_pos
capacity		

The arguments from, to, what, and cell_name all expect strings.

i.e.,

from = "Kalamazoo"

The other arguments expect numbers, i.e.,

x_pos = 237

If an argument is excluded, the parser will fill in a default value for it.

5.3.5 What the Arguments are For

from Text specifying who the predecessor of a current cell is, so a buffer connection can be made between current cell's input, and **from** cell's output.

to Text specifying which cell comes next, so current cell's output can be connected to the **to** cell's input.

what When specifying a buffer connection, the **what** argument tells what the buffer will hold.

cell_name Text name of a cell.

thru_put Throughput for a cell.

num_in Number of items processed at a time by a cell.

num_out Number of items output by a cell when it is done processing.

x_pos x co-ordinate, in pixels, of a cell. When used with size_window, it refers to window width in pixels.

y_pos y co-ordinate, in pixels, of a cell. When used with size_window, it refers to window height in pixels.

capacity Number of items a buffer can hold.

Not all argument types are relevant for all prefixes. In fact, the parser only allows certain arguments for each prefix:

Prefix	Allowed Arguments
connect_output	from, to, what, capacity
connect_input	from, to, what, capacity
sequential_machine	cell_name, thru_put, num_in, num_out, x_pos, y_pos
sink	cell_name, thru_put, num_in, x_pos, y_pos
source	cell_name, thru_put, num_out, x_pos, y_pos
parallel_machine	cell_name, thru_put, x_pos, y_pos
hetero_machine	cell_name, thru_put
size_window	x_pos, y_pos
process_machine	cell_name, num_in, num_out, x_pos, y_pos

5.3.6 A Quick Glossary of Terms

Cell - In the simulation, a cell corresponds to some sort of real life machine, for example a sewing machine or a cloth cutting machine.

Buffer - A buffer corresponds to anything in between two cells. In real life, this could be a conveyor belt, or simply pieces of fabric stacked on the floor.

Node - This can refer either generically to a member of a list, or to the class. When referring to the class, node will be boldfaced.

Simulation Engine - The part of the program that actually runs the simulation. For all practical purposes, this is the set of files with the prefix **vme**, i.e., **vme_cell.c**.

Simulation Object - Anything you see on the screen is a simulation object. Things that look like sewing machines or trucks, for example, are cells. The bar graphs, or thermometers, are buffers.

Widget - The X-windows name for any of the objects you see on the screen. Each cell and buffer has an associated Widget, the Widget being the graphic representation (for instance the Widget for a sequential machine looks like a sewing machine).

5.4. The VME Completion

The VME described above, essentially presents only the user interface for the modular simulation. No operational rules were incorporated into the simulation, particularly the TSS rules. During the final phase of the project, difficulties in maintaining our programming personnel sent us on a search for an efficient way to finish the VME. This search led us to a company called Imagine That! which produces a powerful and flexible simulation software product called Extend. In addition, this company has a module designed especially for manufacturing which accompanies Extend. Amazingly, Extend is designed and operates in almost the exact manner of our VME design, including a graphical, animated user interface, and hooks which allow I/O from the real components of the manufacturing enterprise. This package was acquired from Imagine That! and in a brief period of time we were able to finish the VME. Our simulation program incorporating the TSS rules is available to anyone who has access to a Macintosh computer and the Extend software.

6.0 Deliverable Demonstrations of the TSS line

6.1 Demonstration at the GTRI AAMTD site

The project team has been very impressed with the work Ms. Carol Ring has done at the demonstration center at Southern Tech in setting up a modular line and training the operators to work on this line in a very productive manner. She has experimented with various groups of sewing stations, various arrangements of the sewing stations, and various garments to be sewn in the line. Her initial work was managed solely by herself, with only unintrusive supervision by Mr. Bill Cameron.

We proposed to Carol the idea of implementing the TSS rules in her line. She was sure it wouldn't work since the line had been set up as "push" line. However, she agreed to retrain the

operators and assist with the experimentation we described in Section 4. After seeing the TSS line in operation, Carol has become an enthusiastic believer in the TSS concept, and has implemented the method on a full-time basis.

On March 18, 1993, the TSS line was part of a demonstration presented by the AMTC to a group of Naval officers from Mantech.

6.2 Demonstration At An Apparel Manufacturing Site

A second demonstration of the VME was held on April 21, 1993, in Raleigh, North Carolina, at Champion Products, Inc., Clayton Plant. We made contact with Champion Products through America's 21st, who introduced Champion to the TSS concept. Champion makes simple athletic apparel such as T-shirts and gym shorts. They will soon introduce their most complicated product, a hooded sweatshirt. They are a division of Sarah Lee, which also own Coach Leatherwear, which factory we visited previously.

Our hosts for the demonstration were Mr. Mitchell Johnson, Plant Manager, and Mr. Tom Tanner, Division Training Manager. Both were very helpful, smart, and thoughtful. Champion has about 50 TSS teams of 2-5 workers per team. This plant receives cut cloth from another facility, then sews, and boxes the product. Labelling and packaging is done elsewhere at the present time. Management's goal is to have 80% of their effort stable, 15% seasonal, and 5% experimental/exceptional.

Champion continues to use some sit-down sewing, but they are trying to phase it out, except for workers who have physical attributes which do not allow them to do stand-up sewing. The sit-down sewing area is a jungle of power cords from the ceiling to the machines. There is a big contrast with the TSS area.

Some characteristics of their system are given in the following paragraphs.

- Direct labor represents only 6-8% of the value of a product.
- TSS uses simpler, less expensive, but more flexible machines. This means that TSS-produced items have a higher percentage of direct labor, but the system is more robust because it is not dependent on very expensive, highly productive, but vulnerable special purpose machines.
- Machine utilization is on the order of 10%! This is because of the average of 2.2 machines per worker. Under the bundle system they had 1.00002 machines per worker. They admit that a large investment in capital is part of the price of a TSS system, but claim that one more than makes it up in other ways.

When Champion starts a team on a new product, they sew "in circles" until everyone has experience at all stations. Then management and workers together decide the sequence of workers. They do NOT define zones, (and may have been the ones to originally thought of doing away with them).

- At first, workers stayed within their zones because they wanted to concentrate only on "their" work, not to avoid the work of others, but to specialize on what they were best at (and consequently so they could make more money). Management had to require the team to change the sequence occasionally to maintain cross-training. Without cross-training, productivity crashed every time a new product was introduced or other changes occurred.
- When first setting up TSS, Champion assigned the older and slower workers to the end of the line since the work included inspection and packaging, which was thought to be less demanding than sewing. Productivity took a nose-dive. They now try to put the team leader at the end of the line (closer to the finished product). Their metaphor is that the last worker is the "locomotive that pulls the train."

There are certain incentives which Champion uses to promote the TSS system to their operators.

- Workers are not allowed to "clock out" when their machine is down. Instead it counts against their productivity and pay. This gives workers an incentive to maintain the equipment. The workers also apply considerable pressure on the repairman to come running when something goes wrong. If the machine is not fixed within 15 minutes, it is replaced. Champion maintains about $100,000 worth of backup machines.
- Champion sells TSS as a healthier way of to sew. They claim that repetitive motion injuries, such as carpal tunnel syndrome, are way down under TSS.
- Management guarantees a certain base pay (in the neighborhood of ($5.50). They are expected to earn about $6.25 by meeting standards (which increase each week over a six week period). Each team can increase its base pay by earning up to four "flags," which are awarded for 0 injuries, 100% quality, 100% attendance, and one other criterion. Each worker of any team that has won all four flags during the preceding week gets an addition $0.50 per hour.
- Apparently under the bundle system there are a few superstars earning as much as $16.00 per hour, but with most far below that. Such disparity is not allowed under the TSS

system because it means that something is out of balance. By giving the teams incentives rather than individuals, balance is maintained.

Our impression of their operation is that they run the most effective of the TSS operations we have seen so far. They are devoted to TSS, but have renamed it CSS for *Champion Sewing System*. They think the main advantages of the TSS system over other unit production systems are flexibility and teamwork.

7.0 Summary and Conclusions

In this project we have examined implementation methodologies for the modular manufacturing concept in the apparel industry. We began by gathering background information from the field and from the literature with regard to both the manufacture of apparel and the notion of flexible work groups. We have taken two different views of how such a method could be used, and carefully analyzed both. This work has shown that the idea of organizing the manufacturing floor into teams of operators is more than just a *modular*; it can bring a great deal of *flexibility* to the sewing room.

In the first view of flexible work groups, we looked at the traditional view of the modular group: a small group of operators, some of whom are cross-trained on several operations in the group and can move around in the group to alleviate bottlenecks. We examined how to best coordinate the efforts of such a flexible work group. The solution method for this problem involves a maxflow-matching heuristic. The maxflow portion allows us to find very good throughput rates while the matching portion determines the dynamic worker assignments that occur during the production cycle. All this can be done in real time so that supervisors could learn to use a computerized decision support system which would have this methodology imbedded.

In the second view of flexible work groups, we examined the organization and operation of the line according to methods derived from a commercially marketed manufacturing method known as TSS. We have shown that such a line is *effective* in achieving maximum production rate, the TSS rule is *simple* for the operators to learn, data requirements are *sparse* in that only relative speeds of the workers is needed, and that it is *adaptive* since the line configures itself without management intervention.

Beyond the analytical work in this project, we also developed a tool that has proved invaluable to our work. This tool is called a *virtual manufacturing enterprise* (VME). Although the conceptual idea involves an entire manufacturing enterprise, including suppliers and

distributors, our tool has been developed solely for the flexible work group. It consists of a set of software modules which work together to emulate the actual functioning of a flexible work group. This tool has provided many insights that have led to the results reported in this document. The VME can be invaluable on the factory floor for the facility planners who are concerned with productivity.

In conclusion, we have developed methods for implementing the modular concept which are effective and are themselves flexible. We feel that we have accomplished our goal of adding flexibility to the manufacture of apparel. Our work has provided simple and efficient means for doing effectively what so many apparel manufacturers are struggling to do on the factory floor. We look forward to implementing these ideas in actual factories.

8.0 References

Bollabas, B. (1990), *Linear Analysis*. Cambridge University Press.

Buzacott J. A. (1990), "Abandoning the moving assembly line; models of human operators and job sequencing," *Int. J. Prod. Res.* Vol 28, No. 5, pp 821-839.

Chvatal, Vasek (1983), *Linear Programming*. W. H. Freeman and Company, New York.

Devaney, R. L. (1989), *An Introduction to Chaotic Dynamical Systems, 2nd ed.*, Addison-Wesley, Reading, Mass.

Feller, William (1968), *Introduction to Probability Theory and Its Applications*. John Wiley and Sons, Inc., New York.

Jayaraman, S. and Malhotra, R., (October,1992), "Apparel Manufacturing Architecture [Version 1.0], Volume I: The Function and Synamics Models," SJ-TR-ARCH-9210, Georgia Institute of Technology, Atlanta, Georgia.

Lawler, Eugene L (1976), *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, New York 1976.

Nemhauser, George L. and Laurence A. Wolsey (1988), Integer and Combinatorial Optimization. John Wiley and Sons, Inc., New York.

Ostolaza J., J. McClain, and J. Thomas (1990), "The use of dynamic (state-dependent) assembly line balancing to improve throughput," *J. Mfg. Oper. Mgt.*, Vol. 3, pp. 105-133.

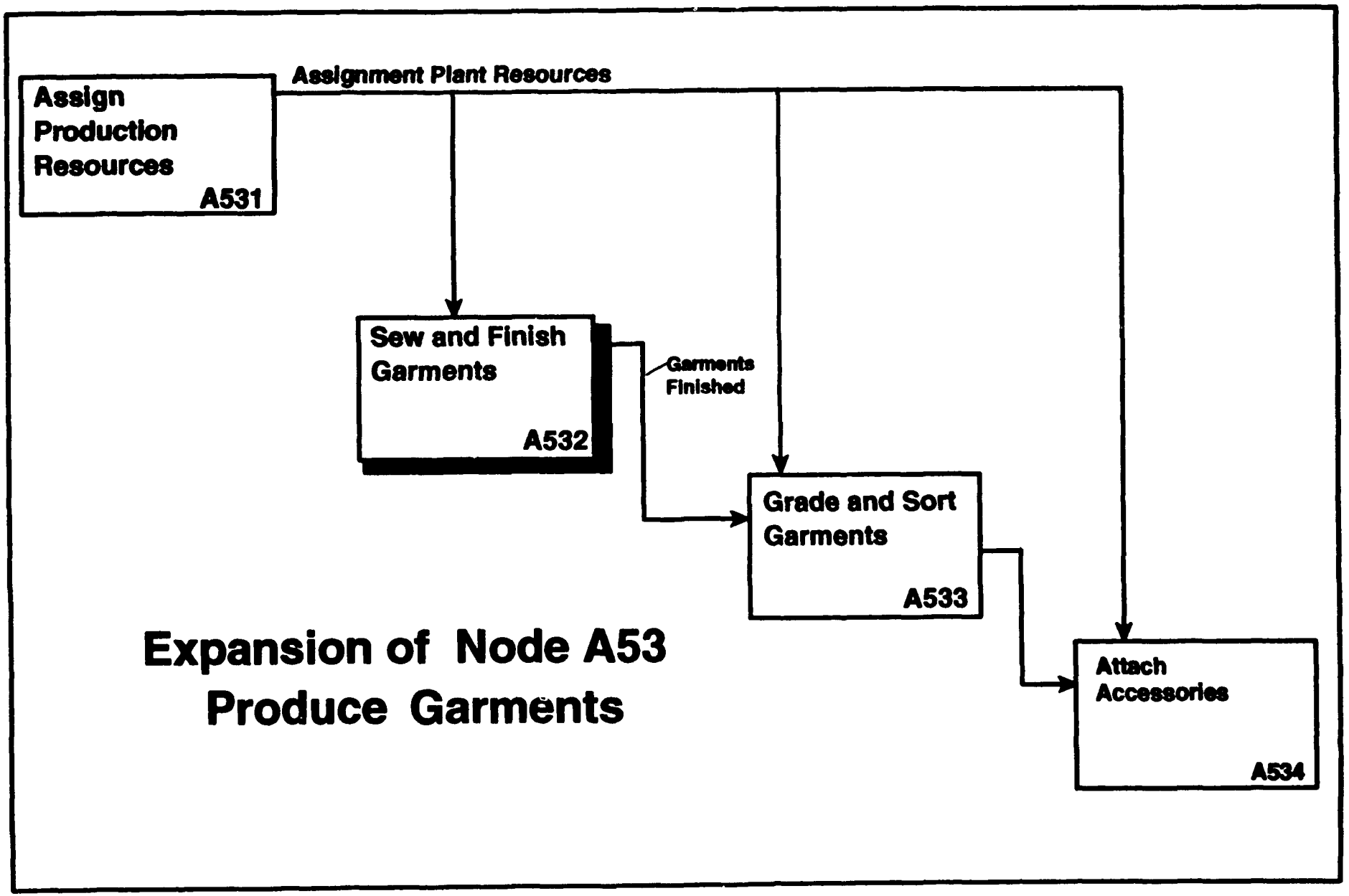
Ross, Sheldon M. (1980), *Introduction to Probability Models*. Academic Press, New York.

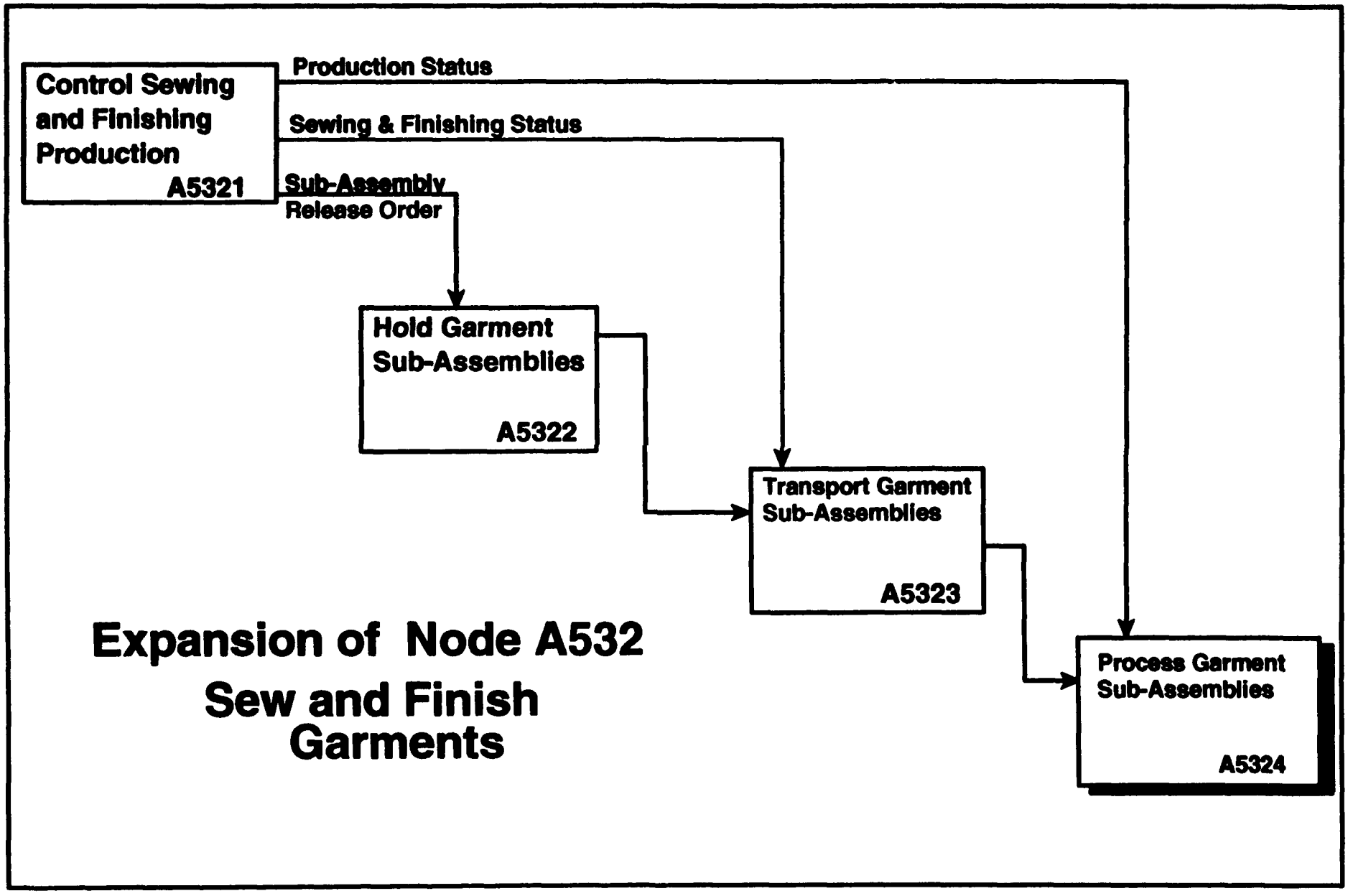
Schroer, B. J., J. Wang, and M. C. Ziemke (1991), "A look at TSS through simulation," *Bobbin* magazine, July, pp. 114-119.

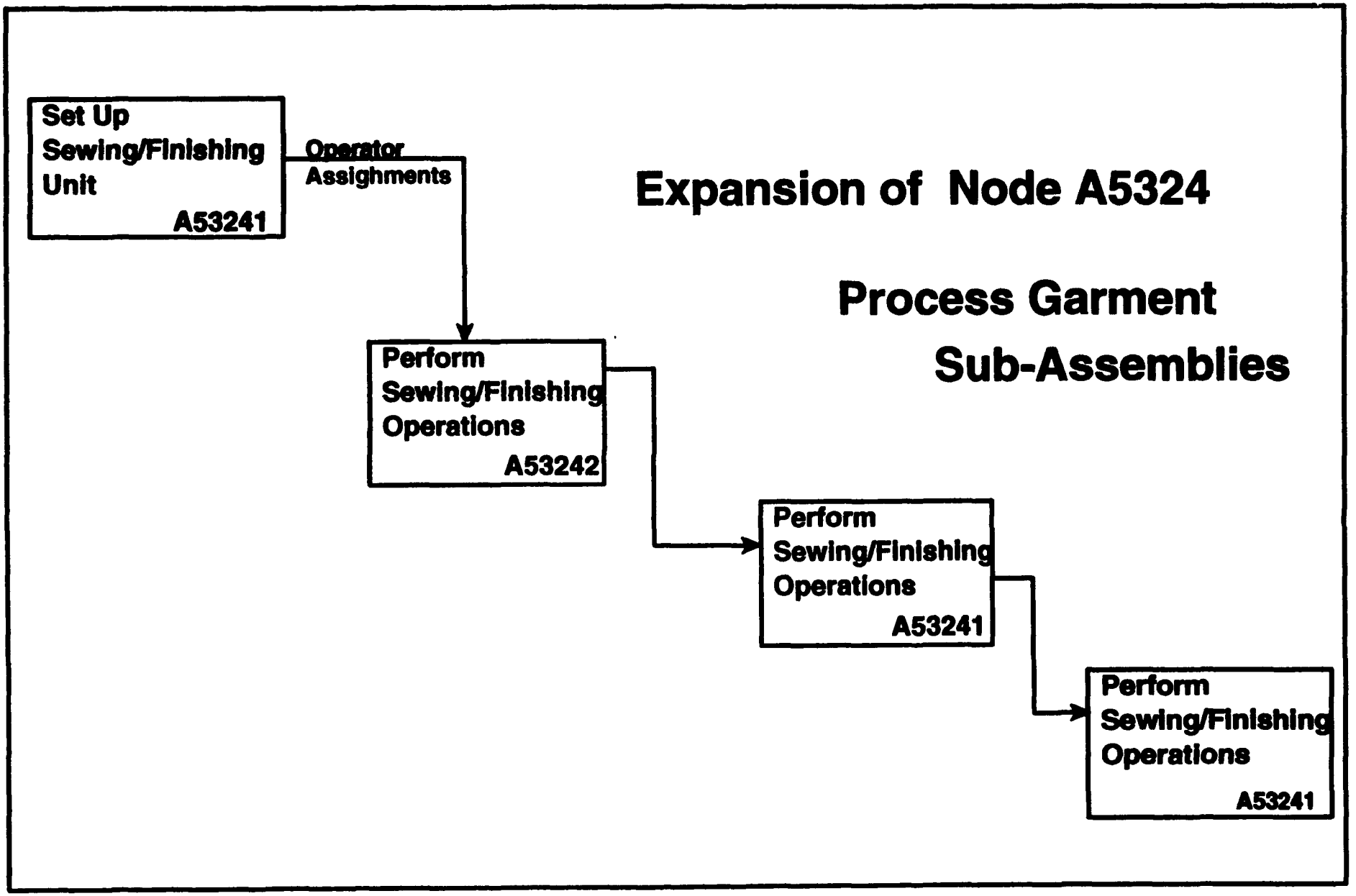
Smart, D. R. (1974), *Fixed Point Theorems*, Cambridge University Press.

Wang, Jian, M. Carl Ziemke, and Terri Adams (1990). Simulation Model of the Yoyota Modular Manufacturing System. Technical Report 90-44, University of Alabama in Huntsville, Industrial and Systems Engineering Department.

Appendix A


Assign Production Resources
A531
Assignment Plant Resources
Sew and Finish Garments
A532
Garments Finished
Grade and Sort Garments
A533
Attach Accessories
A534
Expansion of Node A53
Produce Garments


Control Sewing and Finishing Production
A5321
Production Status
Sewing & Finishing Status
Sub-Assembly Release Order
Hold Garment Sub-Assemblies
A5322
Transport Garment Sub-Assemblies
A5323
Process Garment Sub-Assemblies
A5324
Expansion of Node A532
Sew and Finish Garments


Set Up Sewing/Finishing Unit
A53241
Operator Assighments
Expansion of Node A5324
Process Garment Sub-Assemblies
Perform Sewing/Finishing Operations
A53242
Perform Sewing/Finishing Operations
A53241
Perform Sewing/Finishing Operations
A53241